09980761 .041502

JC14 Rec'd PCT/PTO 26 NOV 2001

FORM PTO-1390 (REV. 9-2001) U S DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

TRANSMITTAL LETTER TO THE UNITED STATES DESIGNATED/ELECTED OFFICE (DO/EO/US) CONCERNING A FILING UNDER 35 U.S.C. 371

ATTORNEY 'S DOCKET NUMBER: 10003628-2

U S APPLICATION NO (If known, see 37 CFR 1 5): 09/980761

INTERNATIONAL APPLICATION NO.	INTERNATIONAL FILING DATE	PRIORITY DATE CLAIMED
PCT/US00/14250	24 May 2000	24 May 1999

TITLE OF INVENTION

RELIABLE MULTI - UNICAST

APPLICANT(S) FOR DO/EO/US

KRAUSE, Michael; RECIO, Renato, John

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. [x] This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371
2. [ ] This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.
3. [ ] This is an express request to begin national examination procedures (35 U.S.C. 371(f)). The submission must include items (5), (6), (9) and (21) indicated below.
4. [ ] The US has been elected by the expiration of 19 months from the priority date (Article 31).
5. [x] A copy of the International Application as filed (35 U S.C 371(c)(2))
   a. [ ] is attached hereto (required only if not communicated by the International Bureau)
   b. [ ] has been communicated by the International Bureau.
   c. [x] is not required, as the application was filed in the United States Receiving Office (RO/US), but is attached hereto
6. [ ] An English language translation of the International Application as filed (35 U.S C. 371(c)(2)).
   a. [ ] is attached hereto.
   b. [ ] has been previously submitted under 35 U.S.C. 154(d)(4).
7. [ ] Amendments to the claims of the International Aplication under PCT Article 19 (35 U.S.C. 371(c)(3))
   a. [ ] are attached hereto (required only if not communicated by the International Bureau).
   b. [ ] have been communicated by the International Bureau.
   c. [ ] have not been made, however, the time limit for making such amendments has NOT expired.
   d. [ ] have not been made and will not be made
8. [ ] An English language translation of the amendments to the claims under PCT Article 19 (35 U.S.C. 371 (c)(3))
9. [ ] An oath or declaration of the inventor(s) (35 U S.C. 371(c)(4)).
10. [ ] An English lanugage translation of the annexes of the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371(c)(5)).

Items 11 to 20 below concern document(s) or information included:

11. [ ] An Information Disclosure Statement under 37 CFR 1.97 and 1.98.
12. [ ] An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3 28 and 3.31 is included.
13. [x] A FIRST preliminary amendment.
14. [ ] A SECOND or SUBSEQUENT preliminary amendment.
15. [ ] A substitute specification.
16. [ ] A change of power of attorney and/or address letter.
17. [ ] A computer-readable form of the sequence listing in accordance with PCT Rule 13ter.2 and 35 U.S.C. 1.821 - 1.825
18. [ ] A second copy of the published international application under 35 U.S.C. 154(d)(4).
19. [ ] A second copy of the English language translation of the international application under 35 U S C. 154(d)(4).
20. [ ] Other items or information:

U.S. APPLICATION NO (if known, see 37 CFR 1.5)	INTERNATIONAL APPLICATION NO	ATTORNEY'S DOCKET NUMBER
09/980761	PCT/US00/14250	10003628-2

21. [X] The following fees are submitted:	CALCULATIONS PTO USE ONLY	
BASIC NATIONAL FEE (37 CFR 1.492 (a) (1) - (5)):		
Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO and International Search Report not prepared by the EPO or JPO **$1040.00**		
International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO **$890.00**		
International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO **$740.00**		
International preliminary examination fee (37 CFR 1.482) paid to USPTO but all claims did not satisfy provisions of PCT Article 33(1)-(4) **$710.00**		
International preliminary examination fee (37 CFR 1.482) paid to USPTO and all claims satisfied provisions of PCT Article 33(1)-(4) **$100.00**		
ENTER APPROPRIATE BASIC FEE AMOUNT =	$710.00	
Surcharge of **$130.00** for furnishing the oath or declaration later than [ ] 20 [X] 30 months from the earliest claimed priority date (37 CFR 1.492(e))	$130.00	

CLAIMS	NUMBER FILED	NUMBER EXTRA	RATE	$
Total claims	46 - 20 =	26	x **$18.00**	$468.00
Independent claims	2 - 3 =	0	x **$84.00**	$
MULTIPLE DEPENDENT CLAIM(S) (if applicable)			+ **$280.00**	$

TOTAL OF ABOVE CALCULATIONS =	$1308.00	
[ ] Applicant claims small entity status. See 37 CFR 1.27 The fees indicated above are reduced by 1/2. +	$	
SUBTOTAL =	$	
Processing fee of **$130.00** for furnishing the English translation later than [ ] 20 [ ] 30 months from the earliest claimed priority date (37 CFR 1.492(f)).	$	
TOTAL NATIONAL FEE =	$1308.00	
Fee for recording the enclosed assignment (37 CFR 1.21(h)) The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31). **$40.00** per property +	$	
TOTAL FEES ENCLOSED =	$1308.00	
	Amount to be refunded:	$
	charged:	$

a. [ ] A check in the amount of $ _____ to cover the above fees is enclosed.

b. [X] Please charge my Deposit Account No. 08-2025 in the amount of $ 1308.00 to cover the above fees. A duplicate copy of this sheet is enclosed

c. [X] The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. 08-2025 . A duplicate copy of this sheet is enclosed.

d. [ ] Fees are to be charged to a credit card. **WARNING:** Information on this form may become public. **Credit card information should not be included on this form.** Provide credit card information and authorization on PTO-2038.

NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137 (a) or (b)) must be filed and granted to restore the application to pending status.

SEND ALL CORRESPONDENCE TO

PATRICK BILLIG
HEWLETT-PACKARD COMPANY
INTELLECTUAL PROPERTY ADMINISTRATION
P.O. Box 272400
3404 E. Harmony Road, M/S 35
Fort Collins, CO 80527-2400

SIGNATURE

Patrick G. Billig
NAME

38,080
REGISTRATION NUMBER

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: Michael R. Krause et al.

Filed: Herewith

Docket No.: 10003628-2

Title: RELIABLE MULTI-UNICAST

Commissioner for Patents
Washington, D.C. 20231

PRELIMINARY AMENDMENT

Sir/Madam:

This Preliminary Amendment modifies the Utility Patent Application filed herewith. Please amend the above-identified application as follows:

IN THE CLAIMS

Please cancel claim 1 without prejudice.

Please add claims 2-47 as follows:

2. A distributed computer system comprising:

a source endnode participating in a multicast group and including:

a source process which produces message data;

a send work queue having work queue elements that describe the message data for multicasting;

multiple destination endnodes participating in the multicast group, each destination endnode including:

a destination process; and

a receive work queue having work queue elements that describe where to place incoming message data;

communication fabric providing communication betw multiple destination endnodes; and

multiple end-to-end contexts, each end-to-end contex information at the source node and a portion storing state inf of the destination endnodes to ensure the reception and seque from the source endnode to the corresponding one of the des reliable multicast comprises a series of replicated unicasts of

Not used to calculate claim fee. Does not address the article 34 amendments in specification
09 Feb 02
C. Burt

work queue and each of the end-to-end contexts portions at the source endnode to the receive work queue and the corresponding end-to-end context portion at each of the destination endnodes.

3. The distributed computer system of claim 2 wherein the source endnode includes a network interface controller which packetizes the message data into frames.

4. The distributed computer system of claim 3 wherein the destination endnodes each include a network interface controller which acknowledges receipt of frames multicast from the source endnode.

5. The distributed computer system of claim 4 wherein the network interface controller and the end-to-end context portion in each destination endnode ensure strong ordering of received frames multicast from the source endnode, such that the frames are received in a same defined order as transmitted from the source endnode.

6. The distributed computer system of claim 4 wherein the source endnode retransmits frames that are not successively acknowledged in the reliable multicast.

7. The distributed computer system of claim 3 wherein the network interface controller in the source endnode includes hardware which replicates frames to be provided in the series of unicasts.

8. The distributed computer system of claim 2 wherein the source endnode includes software verbs which perform the series of unicasts as a series of individual sequenced message send operations.

9. The distributed computer system of claim 2 wherein changes in composition of the endnodes participating in the multicast group are communicated to all endnodes participating in the multicast group.

10. The distributed computer system of claim 2 wherein the source endnode and each destination endnode maintains a list of destination addresses for all other endnodes participating in the multicast group.

11. The distributed computer system of claim 4 wherein the network interface controller in each destination endnode generates cumulative acknowledgments.

12. The distributed computer system of claim 4 wherein the network interface controller in each destination endnode generates acknowledgments on a per frame basis.

13. The distributed computer system of claim 4 the network interface controller of the source endnode includes a completion processing unit which gathers acknowledgements from the destination endnodes and completes frame operation by informing the source process of an operation status of multicast frames.

14. The distributed computer system of claim 13 wherein the source endnode further comprises:

a completion queue containing information related to completed work queue elements, wherein the completion processing unit communicates with the source process via the completion queue.

15. The distributed computer system of claim 13 wherein the completion processing unit informs the source process which destination processes, if any, did not receive multicast frames.

16. The distributed computer system of claim 13 wherein the completion processing unit includes an acknowledgement counter which counts acknowledgements received from the corresponding destination endnodes in the multicast group indicating that the corresponding destination endnode has received a frame multicast from the source endnode.

17. The distributed computer system of claim 16 wherein completion processing unit generates a completion event to the source process when the acknowledgement counter

indicates that a predetermined percentage of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

18. The distributed computer system of claim 16 wherein completion processing unit generates a completion event to the source process when the acknowledgement counter indicates that all of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

19. The distributed computer system of claim 13 wherein the completion processing unit includes a bit-mask array which assigns a unique bit for each destination endnode in the multicast group and clears each bit as a corresponding acknowledgment is received from the corresponding destination endnode in the multicast group indicating that the corresponding destination endnode has received a frame multicast from the source endnode.

20. The distributed computer system of claim 19 wherein the completion processing unit generates a completion event to the source process when the bit-mask array has a predetermined percentage of bits cleared in the bit-mask array indicating that that a predetermined percentage of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

21. The distributed computer system of claim 19 wherein the completion processing unit generates a completion event to the source process when the bit-mask array has all bits cleared in the bit-mask array indicating that that all of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

22 The distributed computer system of claim 13 wherein the completion processing unit includes a timing window, wherein expiring of the timing window without necessary conditions for a completion event for a corresponding multicast frame occurring indicates that any missing acknowledgments are to be tracked and resolved.

23. The distributed computer system of claim 2 wherein a given process joins the multicast group by performing a multicast join operation.

24. The distributed computer system of claim 2 wherein a given process leaves the multicast group by performing a multicast leave operation.

25. A method of multicasting in a distributed computer system including a source endnode participating in a multicast group and multiple destination endnodes participating in the multicast group, the method comprising:

producing message data with a source process at the source endnode;

describing the message data for multicasting with work queue elements in a send work queue at the source endnode;

describing where to place incoming message data with work queue elements in a receive work queue at each of the multiple destination endnodes;

storing in each of multiple end-to-end contexts state information at the source node and state information at a corresponding one of the destination endnodes to ensure the reception and sequencing of message data multicast from the source endnode to the corresponding one of the destination endnodes; and

reliably multicasting data including performing a series of replicated unicasts of message data though the send work queue and each of portions of the end-to-end contexts at the source endnode to the receive work queue and corresponding end-to-end context portions at each of the destination endnodes.

26. The method of claim 25 further comprising:

packetizing, at the source endnode, the message data into frames.

27. The method of claim 26 further comprising:

acknowledging, at each of the destination endnodes, receipt of frames multicast from the source endnode.

28. The method of claim 27 further comprising:

ensuring strong ordering of received frames multicast from the source endnode, such that the frames are received in a same defined order as transmitted from the source endnode.

29. The method of claim 27 further comprising:

retransmitting frames that are not successively acknowledged in the reliable multicast.

30. The method of claim 26 wherein the packetizing the message data into frames includes replicating replicatings frames to be provided in the series of unicasts.

31. The method of claim 25 wherein the series of unicasts are performed as a series of individual sequenced message send operations.

32. The method of claim 25 further comprising:

communicating changes in composition of the endnodes participating in the multicast group to all endnodes participating in the multicast group.

33. The method of claim 25 further comprsing:

maintaining, at the source endnode and each destination endnode, a list of destination addresses for all other endnodes participating in the multicast group.

34. The method of claim 27 wherein the acknowledging, at each of the destination endnodes, includes generating cumulative acknowledgments.

35. The method of claim 27 wherein the acknowledging, at each of the destination endnodes, includes generating acknowledgments on a per frame basis.

36. The method of claim 28 further comprising:

gathering, at the source endnode, acknowledgements from the destination endnodes; and

completing frame operation by informing the source process of an operation status of multicast frames.

37. The method of claim 36 further comprising:

maintaining information related to completed work queue elements in a completion queue; and

communicating with the source process via the completion queue.

38. The method of claim 36 further comrprising:

informing the source process which destination processes, if any, did not receive multicast frames.

39. The method of claim 36 further comprising:

counting acknowledgements received from the corresponding destination endnodes in the multicast group indicating that the corresponding destination endnode has received a frame multicast from the source endnode.

40. The method of claim 39 further comprising:

generating a completion event to the source process when the counted acknowledgements indicate that a predetermined percentage of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

41. The method of claim 39 further comprising:

generating a completion event to the source process when the counted acknowledgements indicate that all of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

42. The method of claim 36 further comprising:

assigning a unique bit in a bit-mask array for each destination endnode in the multicast group; and

clearing each bit in the bit-mask array as a corresponding acknowledgment is received from the corresponding destination endnode in the multicast group indicating that the corresponding destination endnode has received a frame multicast from the source endnode.

43. The method of claim 42 further comprising:

generating a completion event to the source process when a predetermined percentage of bits are cleared in the bit-mask array indicating that that a predetermined percentage of the

destination endnodes in the multicast group have acknowledged the multicast frame has been received.

44. The method of claim 42 further comprising:

generating a completion event to the source process when all bits are cleared in the bit-mask array indicating that that all of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

45. The method of claim 36 further comprising:

maintaining a timing window at the source endnode, wherein expiring of the timing window without necessary conditions for a completion event for a corresponding multicast frame occurring indicates that any missing acknowledgments are to be tracked and resolved.

46. The method of claim 25 further comprising:

performing a multicast join operation to join a given process to the multicast group.

47. The method of claim 25 further comprising:

performing a multicast leave operation to remove a given process from the multicast group.

REMARKS

This Preliminary Amendment modifies the above identified Application filed herewith. In this Preliminary Amendment claim 1 has been canceled without prejudice and claims 2-47 have been substituted therefor. **Applicants request that filing fees be calculated on the basis of entry of this Preliminary Amendment.** Claims 2-47 remain in the application and are presented for consideration and allowance.

It is believed that all claims 2-47 of this application are in condition for allowance. A notice to that effect is respectfully requested.

Preliminary Amendment
Applicant: Michael R. Krause et al.
Filed: Herewith
Docket No.: 10003628-2
Title: RELIABLE MULTI-UNICAST

Any inquiry regarding this Preliminary Amendment should be directed to either Patrick G. Billig, Esq. at Telephone No. (612) 573-2003, Facsimile No. (612) 573-2005 or William J. Streeter, Esq. at Telephone No. (970) 898-3886, Facsimile No. (970) 898-7247. In addition, all correspondence should continue to be directed to the following address:

Hewlett-Packard Company
Intellectual Property Administration
P.O. Box 272400
3404 E. Harmony Road, M/S 35
Fort Collins, Colorado 80527-2400

Respectfully submitted,

Michael R. Krause et al.,

By their attorneys,

DICKE, BILLIG & CZAJA, P.A.
701 Building, Suite 1250
701 Fourth Avenue South
Minneapolis, MN 55415
Telephone: (612) 573-2003
Facsimile: (612) 573-2005

Date: 11-26-01
PGB:cmj

Patrick G. Billig
Reg. No. 38,080

"Express Mail" mailing label number EL874326390US
Date of Deposit November 26, 2001
I hereby certify that this paper or fee is being deposited with the United States Postal Service "Express Mail Post Office to Addressee" services under 37 C F R 1 10 on the date indicated above and is addressed to the Commissioner of Patents and Trademarks, Washington, D.C. 20231

Typed Name of Person Mailing Paper or Fee Christine Docherty

Signature: Christine Docherty

RELIABLE MULTI-UNICAST

Cross-Reference to Related Applications

The Field of the Invention

The present invention generally relates to data processing, and more particularly to a source process communicating with multiple destinations processes via a reliable multicast service.

Background of the Invention

In some conventional data processing systems, a source process can communicate with multiple destination processes via an unreliable multicast service. The source process communicates messages to the destination processes. A message is herein defined to be an application-defined unit of data exchange, which is a primitive unit of communication between cooperating sequential processes. Messages are typically packetized into frames for communication on an underlying communication services/fabrics. A frame is herein defined to be one unit of data encapsulated by a physical network protocol header and/or trailer.

A conventional unreliable multicast service is typically implemented by replicating a frame provided from the source process with a replicater and transmitter component of the underlying communication services/fabric. For example, a switch or a router within the communication services/fabric is typically employed to replicate the frame provided from the source process and to transmit the replicated frame to multiple destination processes which are participating in a target multicast group. The conventional multicast service is designed as an unreliable transport service because of the functional and resource limitations of the traditional replicater and transmitter component. Thus, the conventional unreliable multicast service is only employed in applications which can tolerate frame loss, such as some types of multimedia

applications. Applications which require reliability have to implement reliability features within the application itself because the reliability features are not implemented in the conventional multicast service.

For reasons stated above and for other reasons presented in greater detail in the Description of the Preferred Embodiments section of the present specification, there is a need for a reliable multicast service through which a source process can reliably communicate with multiple destination processes in data processing systems. The reliable multicast service should guarantee that the received order of frames at each of the destination processes in the multicast group is the same as the transmitted order of frames from the source process. In addition, the reliable multicast service should provide the same semantic behaviors as unreliable multicast services to permit existing applications employing conventional unreliable multicast services to employ the reliable multicast service without modifications to the existing application.

Summary of the Invention

The present invention provides a distributed computer system. In one embodiment, the distributed computer system includes a source endnode participating in a multicast group and including a source process which produces message data and a send work queue having work queue elements that describe the message data for multicasting. Multiple destination endnodes are provided participating in a multicast group, each destination endnode including a destination process and a receive work queue having work queue elements that describe where to place incoming message data. A communication fabric provide communication between the source endnode and the mutliple destination endnodes. Multiple end-to-end contexts are provided between the source endnode and each of the destination endnodes, each end-to-end context storing state information at the source node and a corresponding one of the destination endnodes to ensure the reception and sequencing of message data multicast from the source endnode to the multiple destination endnodes thereby permitting reliable multicast service between the source endnode and the multiple destination endnodes.

Brief Description of the Drawings

Figure 1 is a diagram of a distributed computer system for implementing the present invention.

Figure 2 is a diagram of an example host processor node for the computer system of Figure 1.

Figure 3 is a diagram of a portion of a distributed computer system employing a reliable connection service to communicate between distributed processes.

Figure 4 is a diagram of a portion of distributed computer system employing a reliable datagram service to communicate between distributed processes.

Figure 5 is a diagram of an example host processor node for operation in a distributed computer system implementing the present invention.

Figure 6 is a diagram of a portion of a distributed computer system illustrating subnets in the distributed computer system.

Figure 7 is a diagram of a switch for use in a distributed computer system implemented the present invention.

Figure 8 is a diagram of a portion of a distributed computer system.

Figure 9A is a diagram of a work queue element (WQE) for operation in the distributed computer system of Figure 8.

Figure 9B is a diagram of the packetization process of a message created by the WQE of Figure 9A into frames and flits.

Figure 10A is a diagram of a message being transmitted with a reliable transport service illustrating frame transactions.

Figure 10B is a diagram illustrating a reliable transport service illustrating flit transactions associated with the frame transactions of Figure 10A.

Figure 11 is a diagram of a layered architecture for implementing the present invention.

Figure 12 is a diagram of a portion of one embodiment of a distributed computer system employing a unreliable multi-cast service.

Figure 13 is an embodiment of a portion of one embodiment of a distributed computer system employing a reliable multi-unicast service according to the present invention.

Figure 14 is a diagram of one example embodiment of a completion processing unit employing a bit-mask for completing a reliable multi-unicast operation.

Figure 15 is an example embodiment of a completion processing unit employing an acknowledgment counter for completion of a reliable multi-unicast operation.

Description of the Preferred Embodiments

In the following detailed description of the preferred embodiments, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific embodiments in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural or logical changes may be made without departing from the scope of the present invention. The following detailed description, therefore, is not to be taken in a limiting sense, and the scope of the present invention is defined by the appended claims.

One embodiment of the present invention is directed to a reliable multi-unicast service in a data processing system. A source process can reliably communicate with multiple destination process via the reliable multi-unicast service according to the present invention.

An example embodiment of a distributed computer system is illustrated generally at 30 in Figure 1. Distributed computer system 30 is provided merely for illustrative purposes, and the embodiments of the present invention described below can be implemented on computer systems of numerous other types and configurations. For example, computer systems implementing the present invention can range from a small server with one processor and a few input/output (I/O) adapters to massively parallel supercomputer systems with hundreds or thousands of processors and thousands of I/O adapters.

Furthermore, the present invention can be implemented in an infrastructure of remote computer systems connected by an internet or intranet.

Distributed computer system 30 includes a system area network (SAN) 32 which is a high-bandwidth, low-latency network interconnecting nodes within distributed computer system 30. A node is herein defined to be any device attached to one or more links of a network and forming the origin and/or destination of messages within the network. In the example distributed computer system 30, nodes include host processors 34a-34d; redundant array independent disk (RAID) subsystem 33; and I/O adapters 35a and 35b. The nodes illustrated in Figure 1 are for illustrative purposes only, as SAN 32 can connect any number and any type of independent processor nodes, I/O adapter nodes, and I/O device nodes. Any one of the nodes can function as an endnode, which is herein defined to be a device that originates or finally consumes messages or frames in the distributed computer system.

A message is herein defined to be an application-defined unit of data exchange, which is a primitive unit of communication between cooperating sequential processes. A frame is herein defined to be one unit of data encapsulated by a physical network protocol header and/or trailer. The header generally provides control and routing information for directing the frame through SAN 32. The trailer generally contains control and cyclic redundancy check (CRC) data for ensuring packets are not delivered with corrupted contents.

SAN 32 is the communications and management infrastructure supporting both I/O and interprocess communication (IPC) within distributed computer system 30. SAN 32 includes a switched communications fabric (SAN FABRIC) allowing many devices to concurrently transfer data with high-bandwidth and low latency in a secure, remotely managed environment. Endnodes can communicate over multiple ports and utilize multiple paths through the SAN fabric. The multiple ports and paths through SAN 32 can be employed for fault tolerance and increased bandwidth data transfers.

SAN 32 includes switches 36 and routers 38. A switch is herein defined to be a device that connects multiple links 40 together and allows routing of

frames from one link 40 to another link 40 within a subnet using a small header destination ID field. A router is herein defined to be a device that connects multiple links 40 together and is capable of routing frames from one link 40 in a first subnet to another link 40 in a second subnet using a large header destination address or source address.

In one embodiment, a link 40 is a full duplex channel between any two network fabric elements, such as endnodes, switches 36, or routers 38. Example suitable links 40 include, but are not limited to, copper cables, optical cables, and printed circuit copper traces on backplanes and printed circuit boards.

Endnodes, such as host processor endnodes 34 and I/O adapter endnodes 35, generate request frames and return acknowledgment frames. By contrast, switches 36 and routers 38 do not generate and consume frames. Switches 36 and routers 38 simply pass frames along. In the case of switches 36, the frames are passed along unmodified. For routers 38, the network header is modified slightly when the frame is routed. Endnodes, switches 36, and routers 38 are collectively referred to as end stations.

In distributed computer system 30, host processor nodes 34a-34d and RAID subsystem node 33 include at least one system area network interface controller (SANIC) 42. In one embodiment, each SANIC 42 is an endpoint that implements the SAN 32 interface in sufficient detail to source or sink frames transmitted on the SAN fabric. The SANICs 42 provide an interface to the host processors and I/O devices. In one embodiment the SANIC is implemented in hardware. In this SANIC hardware implementation, the SANIC hardware offloads much of CPU and I/O adapter communication overhead. This hardware implementation of the SANIC also permits multiple concurrent communications over a switched network without the traditional overhead associated with communicating protocols. In one embodiment, SAN 32 provides the I/O and IPC clients of distributed computer system 30 zero processor-copy data transfers without involving the operating system kernel process, and employs hardware to provide reliable, fault tolerant communications.

As indicated in Figure 1, router 38 is coupled to wide area network (WAN) and/or local area network (LAN) connections to other hosts or other routers 38.

The host processors 34a-34d include central processing units (CPUs) 44 and memory 46.

I/O adapters 35a and 35b include an I/O adapter backplane 48 and multiple I/O adapter cards 50. Example adapter cards 50 illustrated in Figure 1 include an SCSI adapter card; an adapter card to fiber channel hub and FC-AL devices; an Ethernet adapter card; and a graphics adapter card. Any known type of adapter card can be implemented. I/O adapters 35a and 35b also include a switch 36 in the I/O adapter backplane 48 to couple the adapter cards 50 to the SAN 32 fabric.

RAID subsystem 33 includes a microprocessor 52, memory 54, read/write circuitry 56, and multiple redundant storage disks 58.

SAN 32 handles data communications for I/O and IPC in distributed computer system 30. SAN 32 supports high-bandwidth and scalability required for I/O and also supports the extremely low latency and low CPU overhead required for IPC. User clients can bypass the operating system kernel process and directly access network communication hardware, such as SANICs 42 which enable efficient message passing protocols. SAN 32 is suited to current computing models and is a building block for new forms of I/O and computer cluster communication. SAN 32 allows I/O adapter nodes to communicate among themselves or communicate with any or all of the processor nodes in distributed computer system 30. With an I/O adapter attached to SAN 32, the resulting I/O adapter node has substantially the same communication capability as any processor node in distributed computer system 30.

Channel and Memory Semantics

In one embodiment, SAN 32 supports channel semantics and memory semantics. Channel semantics is sometimes referred to as send/receive or push communication operations, and is the type of communications employed in a

traditional I/O channel where a source device pushes data and a destination device determines the final destination of the data. In channel semantics, the frame transmitted from a source process specifies a destination processes' communication port, but does not specify where in the destination processes' memory space the frame will be written. Thus, in channel semantics, the destination process pre-allocates where to place the transmitted data.

In memory semantics, a source process directly reads or writes the virtual address space of a remote node destination process. The remote destination process need only communicate the location of a buffer for data, and does not need to be involved with the transfer of any data. Thus, in memory semantics, a source process sends a data frame containing the destination buffer memory address of the destination process. In memory semantics, the destination process previously grants permission for the source process to access its memory.

Channel semantics and memory semantics are typically both necessary for I/O and IPC. A typical I/O operation employs a combination of channel and memory semantics. In an illustrative example I/O operation of distributed computer system 30, host processor 34a initiates an I/O operation by using channel semantics to send a disk write command to I/O adapter 35b. I/O adapter 35b examines the command and uses memory semantics to read the data buffer directly from the memory space of host processor 34a. After the data buffer is read, I/O adapter 35b employs channel semantics to push an I/O completion message back to host processor 34a.

In one embodiment, distributed computer system 30 performs operations that employ virtual addresses and virtual memory protection mechanisms to ensure correct and proper access to all memory. In one embodiment, applications running in distributed computed system 30 are not required to use physical addressing for any operations.

Queue Pairs

An example host processor node 34 is generally illustrated in Figure 2. Host processor node 34 includes a process A indicated at 60 and a process B

indicated at 62. Host processor node 34 includes SANIC 42. Host processor node 34 also includes queue pairs (QP's) 64a and 64b which provide communication between process 60 and SANIC 42. Host processor node 34 also includes QP 64c which provides communication between process 62 and SANIC 42. A single SANIC, such as SANIC 42 in a host processor 34, can support thousands of QPs. By contrast, a SAN interface in an I/O adapter 35 typically supports less than ten QPs.

Each QP 64 includes a send work queue 66 and a receive work queue 68. A process, such as processes 60 and 62, calls an operating-system specific programming interface which is herein referred to as verbs, which place work items, referred to as work queue elements (WQEs) onto a QP 64. A WQE is executed by hardware in SANIC 42. SANIC 42 is coupled to SAN 32 via physical link 40. Send work queue 66 contains WQEs that describe data to be transmitted on the SAN 32 fabric. Receive work queue 68 contains WQEs that describe where to place incoming data from the SAN 32 fabric.

Host processor node 34 also includes completion queue 70a interfacing with process 60 and completion queue 70b interfacing with process 62. The completion queues 70 contain information about completed WQEs. The completion queues are employed to create a single point of completion notification for multiple QPs. A completion queue entry is a data structure on a completion queue 70 that describes a completed WQE. The completion queue entry contains sufficient information to determine the QP that holds the completed WQE. A completion queue context is a block of information that contains pointers to, length, and other information needed to manage the individual completion queues.

Example WQEs include work items that initiate data communications employing channel semantics or memory semantics; work items that are instructions to hardware in SANIC 42 to set or alter remote memory access protections; and work items to delay the execution of subsequent WQEs posted in the same send work queue 66.

More specifically, example WQEs supported for send work queues 66 are as follows. A send buffer WQE is a channel semantic operation to push a local buffer to a remote QP's receive buffer. The send buffer WQE includes a gather list to combine several virtual contiguous local buffers into a single message that is pushed to a remote QP's receive buffer. The local buffer virtual addresses are in the address space of the process that created the local QP.

A remote direct memory access (RDMA) read WQE provides a memory semantic operation to read a virtually contiguous buffer on a remote node. The RDMA read WQE reads a virtually contiguous buffer on a remote endnode and writes the data to a virtually contiguous local memory buffer. Similar to the send buffer WQE, the local buffer for the RDMA read WQE is in the address space of the process that created the local QP. The remote buffer is in the virtual address space of the process owning the remote QP targeted by the RDMA read WQE.

A RDMA write WQE provides a memory semantic operation to write a virtually contiguous buffer on a remote node. The RDMA write WQE contains a scatter list of locally virtually contiguous buffers and the virtual address of the remote buffer into which the local buffers are written.

A RDMA FetchOp WQE provides a memory semantic operation to perform an atomic operation on a remote word. The RDMA FetchOp WQE is a combined RDMA read, modify, and RDMA write operation. The RDMA FetchOp WQE can support several read-modify-write operations, such as Compare and Swap if equal.

A bind/unbind remote access key (RKey) WQE provides a command to SANIC hardware to modify the association of a RKey with a local virtually contiguous buffer. The RKey is part of each RDMA access and is used to validate that the remote process has permitted access to the buffer.

A delay WQE provides a command to SANIC hardware to delay processing of the QP's WQEs for a specific time interval. The delay WQE permits a process to meter the flow of operations into the SAN fabric.

In one embodiment, receive queues 68 only support one type of WQE, which is referred to as a receive buffer WQE. The receive buffer WQE provides a channel semantic operation describing a local buffer into which incoming send messages are written. The receive buffer WQE includes a scatter list describing several virtually contiguous local buffers. An incoming send message is written to these buffers. The buffer virtual addresses are in the address space of the process that created the local QP.

For IPC, a user-mode software process transfers data through QPs 64 directly from where the buffer resides in memory. In one embodiment, the transfer through the QPs bypasses the operating system and consumes few host instruction cycles. QPs 64 permit zero processor-copy data transfer with no operating system kernel involvement. The zero processor-copy data transfer provides for efficient support of high-bandwidth and low-latency communication.

Transport Services

When a QP 64 is created, the QP is set to provide a selected type of transport service. In one embodiment, a distributed computer system implementing the present invention supports four types of transport services.

A portion of a distributed computer system employing a reliable connection service to communicate between distributed processes is illustrated generally at 100 in Figure 3. Distributed computer system 100 includes a host processor node 102, a host processor node 104, and a host processor node 106. Host processor node 102 includes a process A indicated at 108. Host processor node 104 includes a process B indicated at 110 and a process C indicated at 112. Host processor node 106 includes a process D indicated at 114.

Host processor node 102 includes a QP 116 having a send work queue 116a and a receive work queue 116b; a QP 118 having a send work queue 118a and receive work queue 118b; and a QP 120 having a send work queue 120a and a receive work queue 120b which facilitate communication to and from process A indicated at 108. Host processor node 104 includes a QP 122 having a send

work queue 122a and receive work queue 122b for facilitating communication to and from process B indicated at 110. Host processor node 104 includes a QP 124 having a send work queue 124a and receive work queue 124b for facilitating communication to and from process C indicated at 112. Host processor node 106 includes a QP 126 having a send work queue 126a and receive work queue 126b for facilitating communication to and from process D indicated at 114.

The reliable connection service of distributed computer system 100 associates a local QP with one and only one remote QP. Thus, QP 116 is connected to QP 122 via a non-sharable resource connection 128 having a non-sharable resource connection 128a from send work queue 116a to receive work queue 122b and a non-sharable resource connection 128b from send work queue 122a to receive work queue 116b. QP 118 is connected to QP 124 via a non-sharable resource connection 130 having a non-sharable resource connection 130a from send work queue 118a to receive work queue 124b and a non-sharable resource connection 130b from send work queue 124a to receive work queue 118b. QP 120 is connected to QP 126 via a non-sharable resource connection 132 having a non-sharable resource connection 132a from send work queue 120a to receive work queue 126b and a non-sharable resource connection 132b from send work queue 126a to receive work queue 120b.

A send buffer WQE placed on one QP in a reliable connection service causes data to be written into the receive buffer of the connected QP. RDMA operations operate on the address space of the connected QP.

The reliable connection service requires a process to create a QP for each process which is to communicate with over the SAN fabric. Thus, if each of N host processor nodes contain M processes, and all M processes on each node wish to communicate with all the processes on all the other nodes, each host processor node requires $M^2 \times (N - 1)$ QPs. Moreover, a process can connect a QP to another QP on the same SANIC.

In one embodiment, the reliable connection service is made reliable because hardware maintains sequence numbers and acknowledges all frame transfers. A combination of hardware and SAN driver software retries any failed

communications. The process client of the QP obtains reliable communications even in the presence of bit errors, receive buffer underruns, and network congestion. If alternative paths exist in the SAN fabric, reliable communications can be maintained even in the presence of failures of fabric switches or links.

In one embodiment, acknowledgements are employed to deliver data reliably across the SAN fabric. In one embodiment, the acknowledgement is not a process level acknowledgment, because the acknowledgment does not validate the receiving process has consumed the data. Rather, the acknowledgment only indicates that the data has reached its destination.

A portion of a distributed computer system employing a reliable datagram service to communicate between distributed processes is illustrated generally at 150 in Figure 4. Distributed computer system 150 includes a host processor node 152, a host processor node 154, and a host processor node 156. Host processor node 152 includes a process A indicated at 158. Host processor node 154 includes a process B indicated at 160 and a process C indicated at 162. Host processor node 156 includes a process D indicated at 164.

Host processor node 152 includes QP 166 having send work queue 166a and receive work queue 166b for facilitating communication to and from process A indicated at 158. Host processor node 154 includes QP 168 having send work queue 168a and receive work queue 168b for facilitating communication from and to process B indicated at 160. Host processor node 154 includes QP 170 having send work queue 170a and receive work queue 170b for facilitating communication from and to process C indicated at 162. Host processor node 156 includes QP 172 having send work queue 172a and receive work queue 172b for facilitating communication from and to process D indicated at 164. In the reliable datagram service implemented in distributed computer system 150, the QPs are coupled in what is referred to as a connectionless transport service.

For example, a reliable datagram service 174 couples QP 166 to QPs 168, 170, and 172. Specifically, reliable datagram service 174 couples send work queue 166a to receive work queues 168b, 170b, and 172b. Reliable

datagram service 174 also couples send work queues 168a, 170a, and 172a to receive work queue 166b.

The reliable datagram service permits a client process of one QP to communicate with any other QP on any other remote node. At a receive work queue, the reliable datagram service permits incoming messages from any send work queue on any other remote node.

In one embodiment, the reliable datagram service employs sequence numbers and acknowledgments associated with each message frame to ensure the same degree of reliability as the reliable connection service. End-to-end (EE) contexts maintain end-to-end specific state to keep track of sequence numbers, acknowledgments, and time-out values. The end-to-end state held in the EE contexts is shared by all the connectionless QPs communicating between a pair of endnodes. Each endnode requires at least one EE context for every endnode it wishes to communicate with in the reliable datagram service (e.g., a given endnode requires at least N EE contexts to be able to have reliable datagram service with N other endnodes).

The reliable datagram service greatly improves scalability because the reliable datagram service is connectionless. Therefore, an endnode with a fixed number of QPs can communicate with far more processes and endnodes with a reliable datagram service than with a reliable connection transport service. For example, if each of N host processor nodes contain M processes, and all M processes on each node wish to communicate with all the processes on all the other nodes, the reliable connection service requires $M^2 \times (N - 1)$ QPs on each node. By comparison, the connectionless reliable datagram service only requires M QPs + (N –1) EE contexts on each node for exactly the same communications.

A third type of transport service for providing communications is a unreliable datagram service. Similar to the reliable datagram service, the unreliable datagram service is connectionless. The unreliable datagram service is employed by management applications to discover and integrate new switches, routers, and endnodes into a given distributed computer system. The unreliable

datagram service does not provide the reliability guarantees of the reliable connection service and the reliable datagram service. The unreliable datagram service accordingly operates with less state information maintained at each endnode.

A fourth type of transport service is referred to as raw datagram service and is technically not a transport service. The raw datagram service permits a QP to send and to receive raw datagram frames. The raw datagram mode of operation of a QP is entirely controlled by software. The raw datagram mode of the QP is primarily intended to allow easy interfacing with traditional internet protocol, version 6 (IPv6) LAN-WAN networks, and further allows the SANIC to be used with full software protocol stacks to access transmission control protocol (TCP), user datagram protocol (UDP), and other standard communication protocols. Essentially, in the raw datagram service, SANIC hardware generates and consumes standard protocols layered on top of IPv6, such as TCP and UDP. The frame header can be mapped directly to and from an IPv6 header. Native IPv6 frames can be bridged into the SAN fabric and delivered directly to a QP to allow a client process to support any transport protocol running on top of IPv6. A client process can register with SANIC hardware in order to direct datagrams for a particular upper level protocol (e.g., TCP and UDP) to a particular QP. SANIC hardware can demultiplex incoming IPv6 streams of datagrams based on a next header field as well as the destination IP address.

SANIC and I/O Adapter Endnodes

An example host processor node is generally illustrated at 200 in Figure 5. Host processor node 200 includes a process A indicated at 202, a process B indicated at 204, and a process C indicated at 206. Host processor 200 includes a SANIC 208 and a SANIC 210. As discussed above, a host processor endnode or an I/O adapter endnode can have one or more SANICs. SANIC 208 includes a SAN link level engine (LLE) 216 for communicating with SAN fabric 224 via link 217 and an LLE 218 for communicating with SAN fabric 224 via link 219.

SANIC 210 includes an LLE 220 for communicating with SAN fabric 224 via link 221 and an LLE 222 for communicating with SAN fabric 224 via link 223. SANIC 208 communicates with process A indicated at 202 via QPs 212a and 212b. SANIC 208 communicates with process B indicated at 204 via QPs 212c-212n. Thus, SANIC 208 includes N QPs for communicating with processes A and B. SANIC 210 includes QPs 214a and 214b for communicating with process B indicated at 204. SANIC 210 includes QPs 214c-214n for communicating with process C indicated at 206. Thus, SANIC 210 includes N QPs for communicating with processes B and C.

An LLE runs link level protocols to couple a given SANIC to the SAN fabric. RDMA traffic generated by a SANIC can simultaneously employ multiple LLEs within the SANIC which permits striping across LLEs. Striping refers to the dynamic sending of frames within a single message to an endnode's QP through multiple fabric paths. Striping across LLEs increases the bandwidth for a single QP as well as provides multiple fault tolerant paths. Striping also decreases the latency for message transfers. In one embodiment, multiple LLEs in a SANIC are not visible to the client process generating message requests. When a host processor includes multiple SANICs, the client process must explicitly move data on the two SANICs in order to gain parallelism. A single QP cannot be shared by SANICS. Instead a QP is owned by one local SANIC.

The following is an example naming scheme for naming and identifying endnodes in one embodiment of a distributed computer system according to the present invention. A host name provides a logical identification for a host node, such as a host processor node or I/O adapter node. The host name identifies the endpoint for messages such that messages are destine for processes residing on an endnode specified by the host name. Thus, there is one host name per node, but a node can have multiple SANICs.

A globally unique ID (GUID) identifies a transport endpoint. A transport endpoint is the device supporting the transport QPs. There is one GUID associated with each SANIC.

A local ID refers to a short address ID used to identify a SANIC within a single subnet. In one example embodiment, a subnet has up 2^{16} endnodes, switches, and routers, and the local ID (LID) is accordingly 16 bits. A source LID (SLID) and a destination LID (DLID) are the source and destination LIDs used in a local network header. A LLE has a single LID associated with the LLE, and the LID is only unique within a given subnet. One or more LIDs can be associated with each SANIC.

An internet protocol (IP) address (e.g., a 128 bit IPv6 ID) addresses a SANIC. The SANIC, however, can have one or more IP addresses associated with the SANIC. The IP address is used in the global network header when routing frames outside of a given subnet. LIDs and IP addresses are network endpoints and are the target of frames routed through the SAN fabric. All IP addresses (e.g., IPv6 addresses) within a subnet share a common set of high order address bits.

In one embodiment, the LLE is not named and is not architecturally visible to a client process. In this embodiment, management software refers to LLEs as an enumerated subset of the SANIC.

Switches and Routers

A portion of a distributed computer system is generally illustrated at 250 in Figure 6. Distributed computer system 250 includes a subnet A indicated at 252 and a subnet B indicated at 254. Subnet A indicated at 252 includes a host processor node 256 and a host processor node 258. Subnet B indicated at 254 includes a host processor node 260 and host processor node 262. Subnet A indicated at 252 includes switches 264a-264c. Subnet B indicated at 254 includes switches 266a-266c. Each subnet within distributed computer system 250 is connected to other subnets with routers. For example, subnet A indicated at 252 includes routers 268a and 268b which are coupled to routers 270a and 270b of subnet B indicated at 254. In one example embodiment, a subnet has up to 2^{16} endnodes, switches, and routers.

A subnet is defined as a group of endnodes and cascaded switches that is managed as a single unit. Typically, a subnet occupies a single geographic or functional area. For example, a single computer system in one room could be defined as a subnet. In one embodiment, the switches in a subnet can perform very fast worm-hole or cut-through routing for messages.

A switch within a subnet examines the DLID that is unique within the subnet to permit the switch to quickly and efficiently route incoming message frames. In one embodiment, the switch is a relatively simple circuit, and is typically implemented as a single integrated circuit. A subnet can have hundreds to thousands of endnodes formed by cascaded switches.

As illustrated in Figure 6, for expansion to much larger systems, subnets are connected with routers, such as routers 268 and 270. The router interprets the IP destination ID (e.g., IPv6 destination ID) and routes the IP like frame.

In one embodiment, switches and routers degrade when links are over utilized. In this embodiment, link level back pressure is used to temporarily slow the flow of data when multiple input frames compete for a common output. However, link or buffer contention does not cause loss of data. In one embodiment, switches, routers, and endnodes employ a link protocol to transfer data. In one embodiment, the link protocol supports an automatic error retry. In this example embodiment, link level acknowledgments detect errors and force retransmission of any data impacted by bit errors. Link-level error recovery greatly reduces the number of data errors that are handled by the end-to-end protocols. In one embodiment, the user client process is not involved with error recovery no matter if the error is detected and corrected by the link level protocol or the end-to-end protocol.

An example embodiment of a switch is generally illustrated at 280 in Figure 7. Each I/O path on a switch or router has an LLE. For example, switch 280 includes LLEs 282a-282h for communicating respectively with links 284a-284h.

The naming scheme for switches and routers is similar to the above-described naming scheme for endnodes. The following is an example switch and

router naming scheme for identifying switches and routers in the SAN fabric. A switch name identifies each switch or group of switches packaged and managed together. Thus, there is a single switch name for each switch or group of switches packaged and managed together.

Each switch or router element has a single unique GUID. Each switch has one or more LIDs and IP addresses (e.g., IPv6 addresses) that are used as an endnode for management frames.

Each LLE is not given an explicit external name in the switch or router. Since links are point-to-point, the other end of the link does not need to address the LLE.

Virtual Lanes

Switches and routers employ multiple virtual lanes within a single physical link. As illustrated in Figure 6, physical links 272 connect endnodes, switches, and routers within a subnet. WAN or LAN connections 274 typically couple routers between subnets. Frames injected into the SAN fabric follow a particular virtual lane from the frame's source to the frame's destination. At any one time, only one virtual lane makes progress on a given physical link. Virtual lanes provide a technique for applying link level flow control to one virtual lane without affecting the other virtual lanes. When a frame on one virtual lane blocks due to contention, quality of service (QoS), or other considerations, a frame on a different virtual lane is allowed to make progress.

Virtual lanes are employed for numerous reasons, some of which are as follows. Virtual lanes provide QoS. In one example embodiment, certain virtual lanes are reserved for high priority or isonchronous traffic to provide QoS.

Virtual lanes provide deadlock avoidance. Virtual lanes allow topologies that contain loops to send frames across all physical links and still be assured the loops won't cause back pressure dependencies that might result in deadlock.

Virtual lanes alleviate head-of-line blocking. With virtual lanes, a blocked frames can pass a temporarily stalled frame that is destined for a different final destination.

In one embodiment, each switch includes its own crossbar switch. In this embodiment, a switch propagates data from only one frame at a time, per virtual lane through its crossbar switch. In another words, on any one virtual lane, a switch propagates a single frame from start to finish. Thus, in this embodiment, frames are not multiplexed together on a single virtual lane.

Paths in SAN fabric

Referring to Figure 6, within a subnet, such as subnet A indicated at 252 or subnet B indicated at 254, a path from a source port to a destination port is determined by the LID of the destination SANIC port. Between subnets, a path is determined by the IP address (e.g., IPv6 address) of the destination SANIC port.

In one embodiment, the paths used by the request frame and the request frame's corresponding positive acknowledgment (ACK) or negative acknowledgment (NAK) frame are not required to be symmetric. In one embodiment employing oblivious routing, switches select an output port based on the DLID. In one embodiment, a switch uses one set of routing decision criteria for all its input ports. In one example embodiment, the routing decision criteria is contained in one routing table. In an alternative embodiment, a switch employs a separate set of criteria for each input port.

Each port on an endnode can have multiple IP addresses. Multiple IP addresses can be used for several reasons, some of which are provided by the following examples. In one embodiment, different IP addresses identify different partitions or services on an endnode. In one embodiment, different IP addresses are used to specify different QoS attributes. In one embodiment, different IP addresses identify different paths through intra-subnet routes.

In one embodiment, each port on an endnode can have multiple LIDs. Multiple LIDs can be used for several reasons some of which are provided by the following examples. In one embodiment, different LIDs identify different partitions or services on an endnode. In one embodiment, different LIDs are

used to specify different QoS attributes. In one embodiment, different LIDs specify different paths through the subnet.

A one-to-one correspondence does not necessarily exist between LIDs and IP addresses, because a SANIC can have more or less LIDs than IP addresses for each port. For SANICs with redundant ports and redundant conductivity to multiple SAN fabrics, SANICs can, but are not required to, use the same LID and IP address on each of its ports.

Data Transactions

Referring to Figure 1, a data transaction in distributed computer system 30 is typically composed of several hardware and software steps. A client process of a data transport service can be a user-mode or a kernel-mode process. The client process accessess SANIC 42 hardware through one or more QPs, such as QPs 64 illustrated in Figure 2. The client process calls an operating-system specific programming interface which is herein referred to as verbs. The software code implementing the verbs intern posts a WQE to the given QP work queue.

There are many possible methods of posting a WQE and there are many possible WQE formats, which allow for various cost/performance design points, but which do not affect interoperability. A user process, however, must communicate to verbs in a well-defined manner, and the format and protocols of data transmitted across the SAN fabric must be sufficiently specified to allow devices to interoperate in a heterogeneous vendor environment.

In one embodiment, SANIC hardware detects WQE posting and accesses the WQE. In this embodiment, the SANIC hardware translates and validates the WQEs virtual addresses and accesses the data. In one embodiment, an outgoing message buffer is split into one or more frames. In one embodiment, the SANIC hardware adds a transport header and a network header to each frame. The transport header includes sequence numbers and other transport information. The network header includes the destination IP address or the DLID or other suitable destination address information. The appropriate local or global

network header is added to a given frame depending on if the destination endnode resides on the local subnet or on a remote subnet.

A frame is a unit of information that is routed through the SAN fabric. The frame is an endnode-to-endnode construct, and is thus created and consumed by endnodes. Switches and routers neither generate nor consume request frames or acknowledgment frames. Instead switches and routers simply move request frames or acknowledgment frames closer to the ultimate destination. Routers, however, modify the frame's network header when the frame crosses a subnet boundary. In traversing a subnet, a single frame stays on a single virtual lane.

When a frame is placed onto a link, the frame is further broken down into flits. A flit is herein defined to be a unit of link-level flow control and is a unit of transfer employed only on a point-to-point link. The flow of flits is subject to the link-level protocol which can perform flow control or retransmission after an error. Thus, flit is a link-level construct that is created at each endnode, switch, or router output port and consumed at each input port. In one embodiment, a flit contains a header with virtual lane error checking information, size information, and reverse channel credit information.

If a reliable transport service is employed, after a request frame reaches its destination endnode, the destination endnode sends an acknowledgment frame back to the sender endnode. The acknowledgment frame permits the requestor to validate that the request frame reached the destination endnode. An acknowledgment frame is sent back to the requestor after each request frame. The requestor can have multiple outstanding requests before it receives any acknowledgments. In one embodiment, the number of multiple outstanding requests is determined when a QP is created.

Example Request and Acknowledgment Transactions

Figures 8, 9A, 9B, 10A, and 10B together illustrate example request and acknowledgment transactions. In Figure 8, a portion of a distributed computer system is generally illustrated at 300. Distributed computer system 300 includes

a host processor node 302 and a host processor node 304. Host processor node 302 includes a SANIC 306. Host processor node 304 includes a SANIC 308. Distributed computer system 300 includes a SAN fabric 309 which includes a switch 310 and a switch 312. SAN fabric 309 includes a link 314 coupling SANIC 306 to switch 310; a link 316 coupling switch 310 to switch 312; and a link 318 coupling SANIC 308 to switch 312.

In the example transactions, host processor node 302 includes a client process A indicated at 320. Host processor node 304 includes a client process B indicated at 322. Client process 320 interacts with SANIC hardware 306 through QP 324. Client process 322 interacts with SANIC hardware 308 through QP 326. QP 324 and 326 are software data structures. QP 324 includes send work queue 324a and receive work queue 324b. QP 326 includes send work queue 326a and receive work queue 326b.

Process 320 initiates a message request by posting WQEs to send queue 324a. Such a WQE is illustrated at 330 in Figure 9A. The message request of client process 320 is referenced by a gather list 332 contained in send WQE 330. Each entry in gather list 332 points to a virtually contiguous buffer in the local memory space containing a part of the message, such as indicated by virtual contiguous buffers 334a-334d, which respectively hold message 0, parts 0, 1, 2, and 3.

Referring to Figure 9B, hardware in SANIC 306 reads WQE 330 and packetizes the message stored in virtual contiguous buffers 334a-334d into frames and flits. As illustrated in Figure 9B, all of message 0, part 0 and a portion of message 0, part 1 are packetized into frame 0, indicated at 336a. The rest of message 0, part 1 and all of message 0, part 2, and all of message 0, part 3 are packetized into frame 1, indicated at 336b. Frame 0 indicated at 336a includes network header 338a and transport header 340a. Frame 1 indicated at 336b includes network header 338b and transport header 340b.

As indicated in Figure 9B, frame 0 indicated at 336a is partitioned into flits 0-3, indicated respectively at 342a-342d. Frame 1 indicated at 336b is

partitioned into flits 4-7 indicated respectively at 342e - 342h. Flits 342a through 342h respectively include flit headers 344a-344h.

Frames are routed through the SAN fabric, and for reliable transfer services, are acknowledged by the final destination endnode. If not successively acknowledged, the frame is retransmitted by the source endnode. Frames are generated by source endnodes and consumed by destination endnodes. The switches and routers in the SAN fabric neither generate nor consume frames.

Flits are the smallest unit of flow control in the network. Flits are generated and consumed at each end of a physical link. Flits are acknowledged at the receiving end of each link and are retransmitted in response to an error.

Referring to Figure 10A, the send request message 0 is transmitted from SANIC 306 in host processor node 302 to SANIC 308 in host processor node 304 as frames 0 indicated at 336a and frame 1 indicated at 336b. ACK frames 346a and 346b, corresponding respectively to request frames 336a and 336b, are transmitted from SANIC 308 in host processor node 304 to SANIC 306 in host processor node 302.

In Figure 10A, message 0 is being transmitted with a reliable transport service. Each request frame is individually acknowledged by the destination endnode (e.g., SANIC 308 in host processor node 304).

Figure 10B illustrates the flits associated with the request frames 336 and acknowledgment frames 346 illustrated in Figure 10A passing between the host processor endnodes 302 and 304 and the switches 310 and 312. As illustrated in Figure 10B, an ACK frame fits inside one flit. In one embodiment, one acknowledgment flit acknowledges several flits.

As illustrated in Figure 10B, flits 342a-h are transmitted from SANIC 306 to switch 310. Switch 310 consumes flits 342a-h at its input port, creates flits 348a-h at its output port corresponding to flits 342a-h, and transmits flits 348a-h to switch 312. Switch 312 consumes flits 348a-h at its input port, creates flits 350a-h at its output port corresponding to flits 348a-h, and transmits flits 350a-h to SANIC 308. SANIC 308 consumes flits 350a-h at its input port. An acknowledgment flit is transmitted from switch 310 to SANIC 306 to

acknowledge the receipt of flits 342a-h. An acknowledgment flit 354 is transmitted from switch 312 to switch 310 to acknowledge the receipt of flits 348a-h. An acknowledgment flit 356 is transmitted from SANIC 308 to switch 312 to acknowledge the receipt of flits 350a-h.

Acknowledgment frame 346a fits inside of flit 358 which is transmitted from SANIC 308 to switch 312. Switch 312 consumes flits 358 at its input port, creates flit 360 corresponding to flit 358 at its output port, and transmits flit 360 to switch 310. Switch 310 consumes flit 360 at its input port, creates flit 362 corresponding to flit 360 at its output port, and transmits flit 362 to SANIC 306. SANIC 306 consumes flit 362 at its input port. Similarly, SANIC 308 transmits acknowledgment frame 346b in flit 364 to switch 312. Switch 312 creates flit 366 corresponding to flit 364, and transmits flit 366 to switch 310. Switch 310 creates flit 368 corresponding to flit 366, and transmits flit 368 to SANIC 306.

Switch 312 acknowledges the receipt of flits 358 and 364 with acknowledgment flit 370, which is transmitted from switch 312 to SANIC 308. Switch 310 acknowledges the receipt of flits 360 and 366 with acknowledgment flit 372, which is transmitted to switch 312. SANIC 306 acknowledges the receipt of flits 362 and 368 with acknowledgment flit 374 which is transmitted to switch 310.

Architecture Layers and Implementation Overview

A host processor endnode and an I/O adapter endnode typically have quite different capabilities. For example, an example host processor endnode might support four ports, hundreds to thousands of QPs, and allow incoming RDMA operations, while an attached I/O adapter endnode might only support one or two ports, tens of QPs, and not allow incoming RDMA operations. A low-end attached I/O adapter alternatively can employ software to handle much of the network and transport layer functionality which is performed in hardware (e.g., by SANIC hardware) at the host processor endnode.

One embodiment of a layered architecture for implementing the present invention is generally illustrated at 400 in diagram form in Figure 11. The

layered architecture diagram of Figure 11 shows the various layers of data communication paths, and organization of data and control information passed between layers.

Host SANIC endnode layers are generally indicated at 402. The host SANIC endnode layers 402 include an upper layer protocol 404; a transport layer 406; a network layer 408; a link layer 410; and a physical layer 412.

Switch or router layers are generally indicated at 414. Switch or router layers 414 include a network layer 416; a link layer 418; and a physical layer 420.

I/O adapter endnode layers are generally indicated at 422. I/O adapter endnode layers 422 include an upper layer protocol 424; a transport layer 426; a network layer 428; a link layer 430; and a physical layer 432.

The layered architecture 400 generally follows an outline of a classical communication stack. The upper layer protocols employ verbs to create messages at the transport layers. The transport layers pass messages to the network layers. The network layers pass frames down to the link layers. The link layers pass flits through physical layers. The physical layers send bits or groups of bits to other physical layers. Similarly, the link layers pass flits to other link layers, and don't have visibility to how the physical layer bit transmission is actually accomplished. The network layers only handle frame routing, without visibility to segmentation and reassembly of frames into flits or transmission between link layers.

Bits or groups of bits are passed between physical layers via links 434. Links 434 can be implemented with printed circuit copper traces, copper cable, optical cable, or with other suitable links.

The upper layer protocol layers are applications or processes which employ the other layers for communicating between endnodes.

The transport layers provide end-to-end message movement. In one embodiment, the transport layers provide four types of transport services as described above which are reliable connection service; reliable datagram service; unreliable datagram service; and raw datagram service.

The network layers perform frame routing through a subnet or multiple subnets to destination endnodes.

The link layers perform flow-controlled, error controlled, and prioritized frame delivery across links.

The physical layers perform technology-dependent bit transmission and reassembly into flits.

Multicast Services

A distributed computer system according to the present invention, such as distributed computer system 30 of Figure 1, supports unreliable multicast and reliable multicast services. In one embodiment, multicast routing and management leverages existing IP-based multicast standards and algorithms. Nevertheless, the following example multicast features are specific to the SAN fabric type architecture of distributed computer system 30 or other such SAN fabric connected distributed computer systems according to the present invention.

In one embodiment, management messages provide multicast group membership information such as additions to a multicast group, deletions from the multicast group, and cast-outs from the multicast group. Management messages also provide route management information. Management messages manage the SAN fabric, such as SAN fabric, 32 of distributed computer system 30. The management messages are typically supported by all endnodes and routing elements primarily through the LLE contained within the endnode and routing elements. Software verbs provide applications and operating system vendors access to management and data transfer operations. Each endnode needs to be able to participate within the multicast group. Other additional operational messages and algorithms are required to implement a viable multicast solution.

In one embodiment, non-fabric management multicast operations are performed via external software support in a fabric manager to provide multicast group management and route updates. For example, a switch or router can

refuse to participate in multicast operations, and an application is informed per the multicast specification being used by the application as to whether a join/leave operation has succeeded.

Applications address endnodes and routing elements using the application's associated IP address (e.g., IPv6 address). Thus, applications are unaware of the corresponding IP address-to-DLID mapping. In one embodiment, applications use a IPv6 multicast standard (e.g., RFC 2375 – IPv6 multicast address assignments) to manage multicast addresses. Thus, the operating system vendor, independent software vendor, and independent hardware vendor can provide the associated IP address-to-DLID mapping services.

Multicast services employ the same route header DLID field as a unicast DLID. Nevertheless, the DLID field is interpreted differently from multicast services by the source, destination, and intermediate routing elements as follows.

The source endnode targets the multicast DLID for a given send operation. The source endnode fills in the frame's DLID field using the same technique as a unicast send operation.

The switch or router receives a frame and performs a route table lookup using the DLID. An example route table implementation includes the encoding of the target port or ports for a given DLID as a bit mask with one bit per port (e.g., an 8-port switch contains an 8-bit field, a 16-port switch contains a 16-bit field). For each indicated output port, a copy of the frame is transmitted.

The destination endnode receives the frame and compares a multicast DLID with a set of DLIDs which it recognizes as its own. If the DLID is not valid, the destination endnode drops the frame, updates the destination node's error statistics, and issues a management message to a designated management entity informing the management entity to update multicast routes to remove the destination endnode as a target for the received DLID.

In one embodiment of reliable multicast, the destination endnode receives a unicast reliable datagram frame for which the destination endnode generates an acknowledgement. For a local route header, the SLID is valid and

is used by the acknowledgment frame. For a global route header, the endnode determines the route to the source endnode using the IP address and generates an acknowledgment frame using a global route header.

Network Multicast Group Service

Network multicast address groups are specified by IP multicast addresses (e.g., IPv6 multicast addresses). Within a given subnet, multicast addresses resolve to a multicast LID (MLID). An MLID can take more than one path through a switch. MLID path updating within the switch is handled by a fabric manager. Once an MLID is assigned to a multicast address, the fabric manager is responsible for updating the routing tables of the switches within a subnet as end stations join and leave a multicast group. The fabric manager employs substantially the same mechanisms as used to notify switches of changes to unicast LID forwarding entries.

For multicasting to extend between subnets, routers must be multicast enabled. The routers must know what multicast addresses are enabled within each subnet attached to a given router. The routers do not need to know how many end stations within an attached subnet belong to a particular multicast group, rather the routers need to know that that at least one node within the subnet belongs to that multicast group. The routers do not have to forward multicasts to each individual end station as is done with a unicast. The routers map the IP address to its MLID. The switch forwarding mechanism propagates the multicast frame throughout the given subnet. Routers must accept frames for every multicast group which is active within the attached subnet so as to be capable of forwarding multicasts throughout other attached subnets. In general, subnet multicast implementation is independent of the mechanisms employed to route multicasts between subnets.

Unreliable Multicast

A portion of one embodiment of a distributed computer system employing a unreliable multicast service is generally illustrated at 500 in Figure

12. Distributed computer system 500 includes a source host processor endnode 502, and destination host processor endnodes 504, 506, and 508. Source host processor endnode 502 includes a source process 510. Destination host processor endnodes 504, 506, 508 respectively include destination processes 512, 514, and 516. Source host processor endnode 502 includes a SANIC 518. Destination host processor endnodes 504, 506, and 508 respectively include SANICs 520, 522, and 524.

Source process 510 communicates with SANIC 526 via QP 526. Destination process 512 communicates with SANIC 520 via QP 528; destination process 514 communicates with SANIC 522 via QP 530; and destination process 516 communicates with SANIC 524 via QP 532. SANIC 518 communicates with a switch 534 via a link 536. Switch 534 includes input ports 540a-540d and output ports 542a-542d. Switch 534 respectively communicates with SANICs 520, 522, and 524 via links 544, 546, and 548. Only one switch 534 is shown in Figure 12 to more clearly illustrate the unreliable multicast operation, but there are typically numerous switches and/or routers to traverse to reach the destination endnodes in a typical unreliable multicast operation.

In an example unreliable multicast send operation, source host processor endnode 502 creates a frame 550 having a multicast DLID. Source host processor endnode 502 sets the DLID in frame 550 to a previously configured multicast DLID. Frame 550 is transmitted from SANIC 518 to input port 540b of switch 534 via link 536.

Switch 534 receives frame 550 and employs the DLID field in frame 550 to index a route lookup table 552 containing DLIDs of destination multicast targets and accordingly targets output ports 542a, 542b, and 542d of switch 534. In the example unreliable multicast send operation, switch 534 replicates frame 550 by replicating flits as they are received to each output port to accordingly produce replicated frames 554a, 554b, and 554c which are respectively provided to output ports 542a, 542b, and 542d. Switch 534 transmits frames 554a-554c respectively to SANICs 520, 522, and 524 via respective links 544, 546, and 548. In one embodiment of unreliable multicast send operation, switch 534

replicates the frame 550 at each output port, but there are many other suitable techniques for replicating the multicast frames.

In the above example unreliable multicast operation, the unreliable multicast service is implemented with switches and routers by sending a frame to a DLID which has been previously configured to represent a given multicast group.

The following are example implementation details of one embodiment of an unreliable multicast service. The destination multicast endnodes do not verify sequence numbers, and therefore, ordering of frames is not preserved. In addition, frames can be dropped by intermediate nodes. The destination multicast endnodes do not generate acknowledgments. The destination multicast endnodes do not forward the frame to associated permissive QPs.

In one embodiment, the unreliable multicast groups are dynamic in composition. Thus, a source multicast endnode does not typically know whether the multicast group contains endnodes which exist on a different subnet. Therefore, in one embodiment, all multicast frames use a global routing header. Within a local subnet, the source multicast endnode encapsulates the global routing header and associated transport header and data within a local routing header. A destination endnode (e.g., a router) discards the local routing header and parses the global routing header to determine what action should be taken. For example, if the destination endnode is a router, the destination endnode discards the local routing header and multicasts the frame per the routing rules of the subnet or subnets attached to the router.

If any endnode drops a multicast frame, that endnode does not forward subsequent flits for the dropped frame. The endnode enters a state in which all subsequent flits targeting the dropped frame are automatically dropped. When a flit arrives which indicates that the flit is the first flit of a new frame, the endnode exits this state and routes/consumes the new frame following the normal operation procedures.

Even though the unreliable multicast operation is assumed to be unreliable, the underlying SAN fabric needs to perform multicast operations at a

reasonable level of success in order for an application to be successful. Therefore, the switches and routers of the SAN fabric preferably provide sufficient resources per virtual lane to accommodate a reasonable number of multicast operations at a given time. The multicast operation resources are preferably separate from unicast resources to avoid creating congestion issues due to resource constraints.

In one embodiment, the unreliable multicast service complies with similar rules to which apply to unreliable datagram messages. For example, in one embodiment, unreliable multicast messages and unreliable datagram messages are both implemented as single frame messages.

Reliable Multi-Unicast

A portion of one embodiment of a distributed computer system employing a reliable multi-unicast service is generally illustrated at 600 in Figure 13. Distributed computer system 600 includes a source host processor endnode 602, and destination host processor endnodes 604, 606, and 608. Source host processor endnode 602 includes a source process 610. Destination host processor endnodes 604, 606, 608 respectively include destination processes 612, 614, and 616. Source host processor endnode 602 includes SANIC 618. Destination host processor endnodes 604, 606, and 608 respectively include SANICs 620, 622, and 624.

SANIC 618 includes LLE 626 and LLE 628. Source process 610 communicates with SANIC 618 via QP 630. SANIC 620 includes LLE 636. Destination process 612 communicates with SANIC 620 via QP 638. SANIC 622 includes LLE 640. Destination process 614 communicates with SANIC 622 via QP 642. SANIC 624 includes LLE 644. Destination process 616 communicates with SANIC 622 via QP 646.

Host processor node 602 includes end-to-end (EE) context 631a corresponding to host processor node 604, and host processor node 604 includes EE context 631b corresponding to host processor node 602. Host processor node 602 includes EE context 632a corresponding to host processor node 606,

and host processor node 606 includes EE context 632b corresponding to host processor node 602. Host processor node 602 includes EE context 634a corresponding to host processor node 608, and host processor node 608 includes EE context 634b corresponding to host processor node 602. The EE contexts are each a special data structure which holds information to ensure the reception and sequencing of frames multicast in the reliable multi-unicast service. The EE contexts are initialized prior to sending messages. An EE context is required for each active virtual lane. Accordingly, the virtual lane as well as the destination information is employed to point to the EE context.

LLE 626 of SANIC 618 communicates with a switch 648 via a link 650. Switch 648 includes input ports 652a-652d and output ports 654a-654d. LLE 628 of SANIC 618 communicates with a switch 658 via a link 660. Switch 658 includes input ports 662a-662d and output ports 664a-664d. Output port 654a of switch 648 communicates with LLE 636 of SANIC 620 via link 656. Output port 664a of switch 658 communicates with LLE 640 of SANIC 622 via link 666. Output port 664b of switch 658 communicates with LLE 644 of SANIC 624 via link 668. Only two switches (i.e., switches 648 and 658) are shown in Figure 13 to more clearly illustrate the reliable multi-unicast operation, but there are typically numerous switches and/or routers to traverse to reach destination endnodes in a typical reliable multi-unicast operation.

In an example reliable multi-unicast send operation, the reliable multicast is implemented as a series of acknowledged unicasts by source host processor endnode 602. In the embodiment illustrated in Figure 13, the reliable multi-unicast is implemented with software verbs as a series of individual sequenced message send operations. In an alternative embodiment, the replication function is performed by hardware in the source endnode of a multicast group.

As illustrated in Figure 13, the software verbs provide replicated sequenced message send operations which are sent through the send work queue of QP 630, and EE contexts 631a, 632a, and 632b. The individual message send through QP 630 and EE context 631a provides a replicated frame 670 on link 650 to input port 652a of switch 648. The individual message send through QP

630 and EE context 632a provides a replicated frame 672 on link 660 to input port 662a of switch 658. The individual message send through QP 630 and EE context 634a provides a replicated frame 674 on link 660 to input port 662a of switch 658.

In the reliable multi-unicast operation, switches and routers view a reliable multicast operation as a normal unicast operation. Therefore, the route lookup tables in the switches and routers are not modified to indicate the associated DLID is participating in a reliable multicast group. Thus, frame 670 is routed through switch 648 to output port 654a, onto link 656 to provide the frame to LLE 636 of SANIC 620. Similarly, frame 672 is routed through switch 658 to output port 664a, onto link 666 to provide the frame to LLE 640 of SANIC 622. Similarly, frame 674 is routed through switch 658 through output port 664b onto link 668 to provide the frame to LLE 644 of SANIC 624.

A reliable multicast group is set up using the same high-level multicast group operations as an unreliable multicast group, but the underlying interpretation and actions taken in the SAN fabric are quite different. For example, a reliable multicast join operation requires each endnode participating in the multicast to know the exact multicast group composition. In one embodiment, each participating endnode maintains a complete list of destination addresses. In one embodiment, each participating endnode establishes reliable datagram end-to-end context to transmit data, and to track and generate acknowledgments for each frame received at the destination endnodes.

The following are example implementation details of one embodiment of a reliable multi-unicast service. The multicast group's composition is communicated to all multicast members within a given multicast group to avoid frame loss. Frames are strongly ordered and are acknowledged at the destination endnode. In one embodiment, the destination endnode validates the SLID, and generates an acknowledgment based on a corresponding valid SLID.

Reliable multi-unicast DLIDs are source endnode and destination endnode concepts. Thus, in a reliable multi-unicast operation, a multicast DLID consumes a DLID from the available DLID space. The multicast DLID for the

reliable multi-unicast is not reflected within any route lookup table contained within a switch or router. If a reliable multi-unicast DLID is used within a frame, an endnode generates a NAK to indicate that the reliable multicast DLID is an invalid DLID.

Although endnode 602 functions as a source endnode and endnodes 604, 606, and 608 operate as destination endnodes for the reliable multi-unicast operation of Figure 13, endnodes 602, 604, 606, and 608 can each be a source or destination endnode depending on the direction of communication for a given reliable multi-unicast operation.

Each of the participating multicast destination endnode's SANIC generate an ACK for each frame it receives. The ACK can be cumulative or on a per frame/flit basis. Completion processing unit 680 of source endnode 602 gathers all of the acknowledgments (e.g., ACKs and NAKs) and completes the frame operation by informing source process 610 of the operation status of a given frame multicast via a completion queue 681. Completion processing unit 680 also informs source process 610 which destination processes, if any, did not receive the multicast frame as a result of an error in the multicast operation.

A given process participates in a reliable multicast group by performing a multicast join operation. When a join operation is successfully completed, the given process can issue a single frame, and the frame is replicated in the source endnode and reliably transmitted to all destination processes participating in the specified multicast group.

A given process leaves a reliable multicast group by performing a multicast leave operation. The multicast leave operation informs all participating multicast process members that the given process leaving the multicast group is no longer a valid destination process for multicast operations within the multicast group. Once participating processes are informed that a given process is leaving the multicast group, the participating processes remaining in the multicast group perform the appropriate resource and in-flight frame recovery operations.

The reliable multi-unicast service preferably has the same operational model as the unreliable multicast service with respect to frame read/write support (e.g., both the reliable multi-unicast service and the unreliable multicast service support writes but neither the reliable multicast service nor the unreliable multicast service support reads). In one embodiment, the reliable multi-unicast service minimally provides the same semantic behavior as conventional unreliable multicast services to permit existing applications employing conventional unreliable multi-unicast services to employ the reliable multicast service without modification to the existing application. Thus, from the application perspective, the reliable multi-unicast service provides the same basic services as the unreliable multicast service except the reliable multi-unicast service is reliable.

In one embodiment, frames are reliably delivered to destination endnodes participating in the multicast group without application modification.

In the reliable multi-unicast service according to the present invention, applications that cannot tolerate frame loss or reordering, can off-load the reliability function including the complexity and the design steps required to provided reliable communication, to the EE contexts and SANICs. The off-loading of the reliability function from the application permits multiple applications to benefit from sharing EE contexts and SANICs. In addition, applications not having to deal with the reliability concerns in the multicast operations have improved performance and quality due to the ability to optimize and leverage the quality across multiple applications. In addition, the reduced complexity of a given application and reduced design steps to create and verify the given application reduce the time-to-market for the given application.

Applications can use both sender-based and receiver based communication techniques with the reliable multi-unicast service. Unreliable multicast services only support receiver-based communication techniques. In one embodiment, processes in a reliable multi-unicast service use the same sender-based or receiver-based communication and memory

management/protection techniques as used by reliable connection services and reliable datagram services.

In the reliable multi-unicast service, memory is protected using techniques such as Hamlyn protection mechanisms to ensure correct access rights (e.g., no access, read access, write access, read/write access, memory address range verification, and the like) are verified before access is granted to a process.

In the reliable multicast service, sender-based communication requires all participating processes to export identical virtual memory windows and to utilize the same protection mechanisms.

The reliable multi-unicast service according to the present invention assures that frames transmitted from the source process to the multicast destination processes are reliable. Thus, a strong ordering mechanism in the EE context and SANIC guarantees that all destination endnodes within the multicast group receive the frames in the same defined order as transmitted from the source endnode. Frames sent by other source endnodes to the multicast group may be interleaved. Strong ordering is only guaranteed on a per source-to-destination EE context resource basis.

A frame is received exactly once at any destination endnode participating within the multicast group. In other words, duplicate copies of frames which may be generated during an error event or a recovery operation are detected and not delivered to the destination endnode participating within the multicast group. In the reliable multi-unicast service, the source endnode is informed of frame transmission completion via acknowledgments. Types of acknowledgements include an ACK which indicates a frame was successively transmitted and received or a NAK which indicates an unrecoverable error was detected either within the frame or in its transmission.

In one embodiment, QoS is implemented for the reliable multi-unicast service. In this embodiment, QoS allows the reliable multi-unicast operations to be prioritized relative to other traffic within the source endnode according to the

QoS service level objectives for the source endnode and the processes executing in the source endnode.

In one embodiment, the source process issues a series of messages unicast to each destination process participating within the target multicast group. Each unicast is acknowledged (e.g., positively with an ACK or negatively with a NAK) to indicate the operation's success or failure. When a source endnode's completion processing unit has received acknowledgments from all, or a predetermined percentage of, destination endnodes in the target multicast group and has completed recovery from recoverable errors, the completion processing unit completes the unit of work multicast operation and, if required based on application parameters, informs the source process of any errors.

In one embodiment, error recovery in the reliable multi-unicast service is implemented by performing a resynchronization event between the source process and impacted destination process of the multicast group. Since an error can involve multiple destination AIs in the reliable multicast, multiple resynchronization operations may need to be implemented before the frame transaction is considered complete in the reliable multi-unicast service.

Example Reliable Multi-Unicast Group Management Operations

In one embodiment, reliable multi-unicast group management operations are performed based on existing multicast group management standards (e.g., Internet Group Management Protocol (IGMP) multicasting standard) to obtain maximum leverage from the existing standards. Nevertheless, the reliable multi-unicast group management operations necessarily vary from the existing multicast group management standards, such as described below. The reliable multicast group membership changes must be tracked. For each membership change, a control management message must be reliably exchanged and managed.

In one embodiment, when a process issues a multicast group join operation, if a participating endnode or the new endnode lacks the resources to

support the new multicast group membership, the reliable multi-unicast operation is failed. If the process is not authorized to participate in the multicast group (e.g., because of improper access rights), the reliable multi-unicast operation is failed. If complete connectivity between all current group members and the new endnode is not possible, the join operation is failed with the appropriate error indication.

In one embodiment, when a process issues a multicast group leave operation, the process is removed as a valid destination process so that subsequent multicast frames will not target the removed process. In-flight frames are marked to indicate that an acknowledgment not being returned for the target process should not be treated as a non-recoverable error. In one embodiment, all multicast group member processes are informed that the leaving process is removed to allow the processes the option of taking appropriate action. In one embodiment, a non-recoverable error which isolates a given process is treated as an unplanned multicast group leave operation.

In one embodiment, for each of the multicast group management operations, the process is explicitly informed or involved in the group membership changes. In another embodiment, the process relegates any involvement in the group membership changes to the associated SANIC and the process itself takes no responsibility for the group membership changes. The choice between these two embodiments is implementation dependent and based on process service objectives. For example, if a process is designed to replicate data to multiple physical sites to ensure data integrity in the event of disaster, the process can decide that the number of sites currently receiving the frame is not relevant and only focus on ensuring that the number is greater than some application-specific minimum. On the other hand, in another example, a reliable multicast process is used to synchronously update multiple sites to ensure that some data lists (e.g., price lists) are consistent and accurate, and the reliable multicast process ensures that the multicast group is correctly created and that all members receive the reliable, synchronous updates.

Reliable multicast groups require complete membership knowledge in order to function. In addition, some operations, such as setting up of memory windows and protection operations associated with sender-based communication, require additional multicast group management operations to be provided. The following are example multicast group management operations which are implemented along with the multicast group join and leave operations to provide a suitable reliable multi-unicast service.

A get attribute operation queries the current attributes of the multicast group, such as current group membership, multicast addressing, and the like.

A set attribute operation sets the multicast group attributes, such as the multicast address, protection services, and the like.

A remove member operation performs an explicit multicast group leave operation without involving the impacted process. In one embodiment, the remove member operation is performed based on service level objectives, process state information (e.g., connectivity lost, process is unresponsive, etc.), protection violations, and the like.

Completion Processing for Reliable Multi-Unicast

Each source endnode's completion processing unit must verify that all, or a predetermined percentage of, destination endnodes within the multicast group reliably receives each frame transmitted from the source endnode. There are a variety of techniques which are suitable to implement the completion processing unit for the reliable multi-unicast service. For example, in one embodiment, the completion processing unit employs an acknowledgement counter which is incremented or decremented with each acknowledgment received. In one form of this embodiment, the completion processing unit generates a completion event to completion queue 681 when the counter indicates that all, or a predetermined percentage of, destination endnodes participating in the multicast group have acknowledged the frame has been received.

In another embodiment, the source endnodes completion processing unit employs a bit-mask array which assigns a unique bit for each destination

endnode participating in the multicast group and clears each bit as each acknowledgment is received. All of these unit of work completion processing unit implementations must assure that the completion processing unit does not generate a completion event to the source process via completion queue 681 until all, or a predetermined percentage of, destination endnodes participating in the multicast group have acknowledged the frame has been received or an unrecoverable error has been detected when processing the frame to a given destination endnode.

In one embodiment, the source completion processing unit includes a timing window. If the timing window expires without the necessary conditions for a completion event occurring, then the missing acknowledgment(s) are tracked and resolved.

One example embodiment of a suitable completion processing unit 680 is generally indicated at 680′ in Figure 14. Completion processing unit 680′ maintains a bit-mask for each in-flight frame in a reliable multi-unicast operation. For example, a frame completion bit-mask is indicated at 682 for frame X and a frame completion bit-mask is indicated at 684 for frame Y.

Each bit in the UW completion bit-mask corresponds to a group member, such as bit 604 corresponding to endnode 604, bit 606 corresponding to endnode 606, and bit 608 corresponding to endnode 608. The state of the bits in the completion bit-mask represent whether the corresponding endnode has acknowledged the frame.

The number of frames which can be tracked is implementation dependent and based upon the resource availability. As acknowledgments are received by source endnode 602, the corresponding bit in the completion bit-mask is cleared. When the completion bit-mask is all cleared (e.g., all zeros), a completion event, for the corresponding frame, is signaled to source process 610 via completion queue 681. To ensure proper ordering, the completion event for frame Y can not be issued until the completion event for frame X has been issued.

Another example suitable embodiment of completion processing unit 680 is generally indicated at 680″ in Figure 15. Completion processing unit 680″

maintains an acknowledgment counter 688 for frame X which is incremented or decremented with each acknowledgment received for frame X. Completion processing unit 680" generates a completion event to completion queue 681 for frame X, when the acknowledgment counter 688 indicates that all three destination endnodes 604, 606, and 608 participating in the multicast group have acknowledged that frame X has been received. Completion processing unit 680" also maintains an acknowledgment counter 690 for frame Y which operates similar to acknowledgment counter 688 to track acknowledgments received corresponding to the multicast frame Y. The number of frames which can be tracked by the acknowledgment counter is implementation dependent and based upon resource availability. To ensure proper ordering, the completion event for frame Y cannot be issued until the completion event for frame X has been issued.

Completion processing unit embodiment 680' and 680" both includes a timing window 686. If timing window 686 expires without the necessary conditions for a completion event for a corresponding frame occurring, then the missing acknowledgments are tracked and resolved.

Although specific embodiments have been illustrated and described herein for purposes of description of the preferred embodiment, it will be appreciated by those of ordinary skill in the art that a wide variety of alternate and/or equivalent implementations calculated to achieve the same purposes may be substituted for the specific embodiments shown and described without departing from the scope of the present invention. Those with skill in the chemical, mechanical, electro-mechanical, electrical, and computer arts will readily appreciate that the present invention may be implemented in a very wide variety of embodiments. This application is intended to cover any adaptations or variations of the preferred embodiments discussed herein. Therefore, it is manifestly intended that this invention be limited only by the claims and the equivalents thereof.

WHAT IS CLAIMED IS:

1. A distributed computer system comprising:

a source endnode participating in a multicast group and including:

a source process which produces message data;

a send work queue having work queue elements that describe the message data for multicasting;

multiple destination endnodes participating in a multicast group, each destination endnode including:

a destination process;

a receive work queue having work queue elements that describe where to place incoming message data;

communication fabric providing communication between the source endnode and the multiple destination endnodes; and

multiple end-to-end contexts, each end-to-end context having a portion storing state information at the source node and a portion storing state information at a corresponding one of the destination endnodes to ensure the reception and sequencing of message data multicast from the source endnode to the corresponding one of the destination endnodes, wherein a reliable multicast comprises a series of replicated unicasts of message data though the send work queue and each of the end-to-end contexts portions at the source endnode to the receive work queue and the corresponding end-to-end context portion at each of the destination endnodes.

2. The distributed computer system of claim 1 wherein the source endnode includes a network interface controller which packetizes the message data into frames.

3. The distributed computer system of claim 2 wherein the destination endnodes each include a network interface controller which acknowledges receipt of frames multicast from the source endnode.

4. The distributed computer system of claim 3 wherein the network interface controller and the end-to-end context portion in each destination

endnode ensure strong ordering of received frames multicast from the source endnode, such that the frames are received in a same defined order as transmitted from the source endnode.

5. The distributed computer system of claim 3 wherein the source endnode retransmits frames that are not successively acknowledged in the reliable multicast.

6. The distributed computer system of claim 2 wherein the network interface controller in the source endnode includes hardware which replicates frames to be provided in the series of unicasts.

7. The distributed computer system of claim 2 wherein the source endnode includes software verbs which perform the series of unicasts as a series of individual sequenced message send operations.

8. The distributed computer system of claim 1 wherein changes in composition of the endnodes participating in the multicast group are communicated to all endnodes participating in the multicast group.

9. The distributed computer system of claim 1 wherein the source endnode and each destination endnode maintains a list of destination addresses for all other endnodes participating in the multicast group.

10. The distributed computer system of claim 3 wherein the network interface controller in each destination endnode generates cumulative acknowledgments.

11. The distributed computer system of claim 3 wherein the network interface controller in each destination endnode generates acknowledgments on a per frame basis.

12. The distributed computer system of claim 3 the network interface controller of the source endnode includes a completion processing unit which

gathers acknowledgements from the destination endnodes and completes frame operation by informing the source process of an operation status of multicast frames.

13. The distributed computer system of claim 12 wherein the source endnode further comprises:

a completion queue containing information related to completed work queue elements, wherein the completion processing unit communicates with the source process via the completion queue.

14. The distributed computer system of claim 12 wherein the completion processing unit informs the source process which destination processes, if any, did not receive multicast frames.

15. The distributed computer system of claim 12 wherein the completion processing unit includes an acknowledgement counter which counts acknowledgements received from the corresponding destination endnodes in the multicast group indicating that the corresponding destination endnode has received a frame multicast from the source endnode.

16. The distributed computer system of claim 15 wherein completion processing unit generates a completion event to the source process when the acknowledgement counter indicates that a predetermined percentage of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

17. The distributed computer system of claim 15 wherein completion processing unit generates a completion event to the source process when the acknowledgement counter indicates that all of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

18. The distributed computer system of claim 12 wherein the completion processing unit includes a bit-mask array which assigns a unique bit for each destination endnode in the multicast group and clears each bit as a corresponding

acknowledgment is received from the corresponding destination endnode in the multicast group indicating that the corresponding destination endnode has received a frame multicast from the source endnode.

19. The distributed computer system of claim 18 wherein the completion processing unit generates a completion event to the source process when the bit-mask array has a predetermined percentage of bits cleared in the bit-mask array indicating that that a predetermined percentage of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

20. The distributed computer system of claim 18 wherein the completion processing unit generates a completion event to the source process when the bit-mask array has all bits cleared in the bit-mask array indicating that that all of the destination endnodes in the multicast group have acknowledged the multicast frame has been received.

21. The distributed computer system of claim 12 wherein the completion processing unit includes a timing window, wherein expiring of the timing window without necessary conditions for a completion event for a corresponding multicast frame occurring indicates that any missing acknowledgments are to be tracked and resolved.

22. The distributed computer system of claim 1 wherein a given process joins the multicast group by performing a multicast join operation.

23. The distributed computer system of claim 1 wherein a given process leaves the multicast group by performing a multicast leave operation.

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 November 2000 (30.11.2000)

PCT

(10) International Publication Number
WO 00/72421 A1

(51) International Patent Classification[7]: H02H 3/05, G06F 15/16

(21) International Application Number: PCT/US00/14250

(22) International Filing Date: 24 May 2000 (24.05.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/135,664 24 May 1999 (24.05.1999) US
60/154,150 15 September 1999 (15.09.1999) US

(71) Applicants *(for all designated States except US)*: **HEWLETT-PACKARD COMPANY** [US/US]; Intellectual Property Administration, 3404 East Harmony Road, M/S 35, P.O. Box 272400, Fort Collins, CO 80527-2400 (US). **IBM CORPORATION** [US/US]; Intellectual Property Administration, 3404 East Harmony Road, M/S 35, P.O. Box 272400, Fort Collins, CO 80527-2400 (US). **COMPAQ COMPUTER CORP.** [US/US]; Intellectual Property Administration, 3404 East Harmony Road, M/S 35, P.O. Box 272400, Fort Collins, CO 80527-2400 (US). **ADAPTEC, INC.** [US/US]; Intellectual Property Administration, 3404 East Harmony Road, M/S 35, P.O. Box 272400, Fort Collins, CO 80527-2400 (US).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **KRAUSE, Michael** [US/US]; 17523 Bear Creek Road, Boulder Creek, CA 95006 (US). **RECIO, Renato, John** [US/US]; 6707 Winnepeg Cove, Austin, TX 78759 (US).

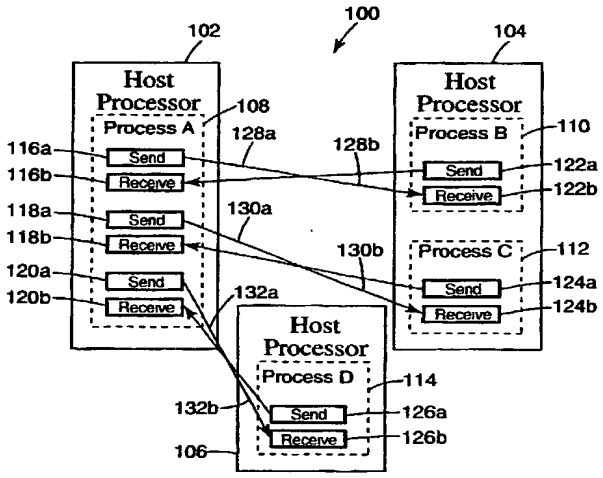
(74) Agents: **BILLIG, Patrick**; Dicke, Billig & Czaja, P.A., Suite 1250, 701 Fourth Avenue South, Minneapolis, MN 55415_ et al. (US).

(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European

[Continued on next page]

(54) Title: RELIABLE MULTI-UNICAST



(57) Abstract: A distributed computer system includes a source endnode (102) participating in a multicast group and including a source process (108) which produces message data and a send work queue (116a) having work queue elements that describe the message data for multicasting. Multiple destination endnodes (104, 106) participate in a multicast group, each destination endnode including a destination process (110, 114) and a receive work queue (122b, 126b) having work queue elements that describe where to place incoming message data. Multiple end-to-end contexts are provided between the source endnode and each of the destination endnodes, each end-to-end context storing state information at the source node (102) and a corresponding one of the destination endnodes (104, 106) to ensure the reception and sequencing of message data multicast from the source endnode to the multiple destination endnodes thereby permitting reliable multicast service between the source endnode and multiple destination endnodes.

WO 00/72421 A1

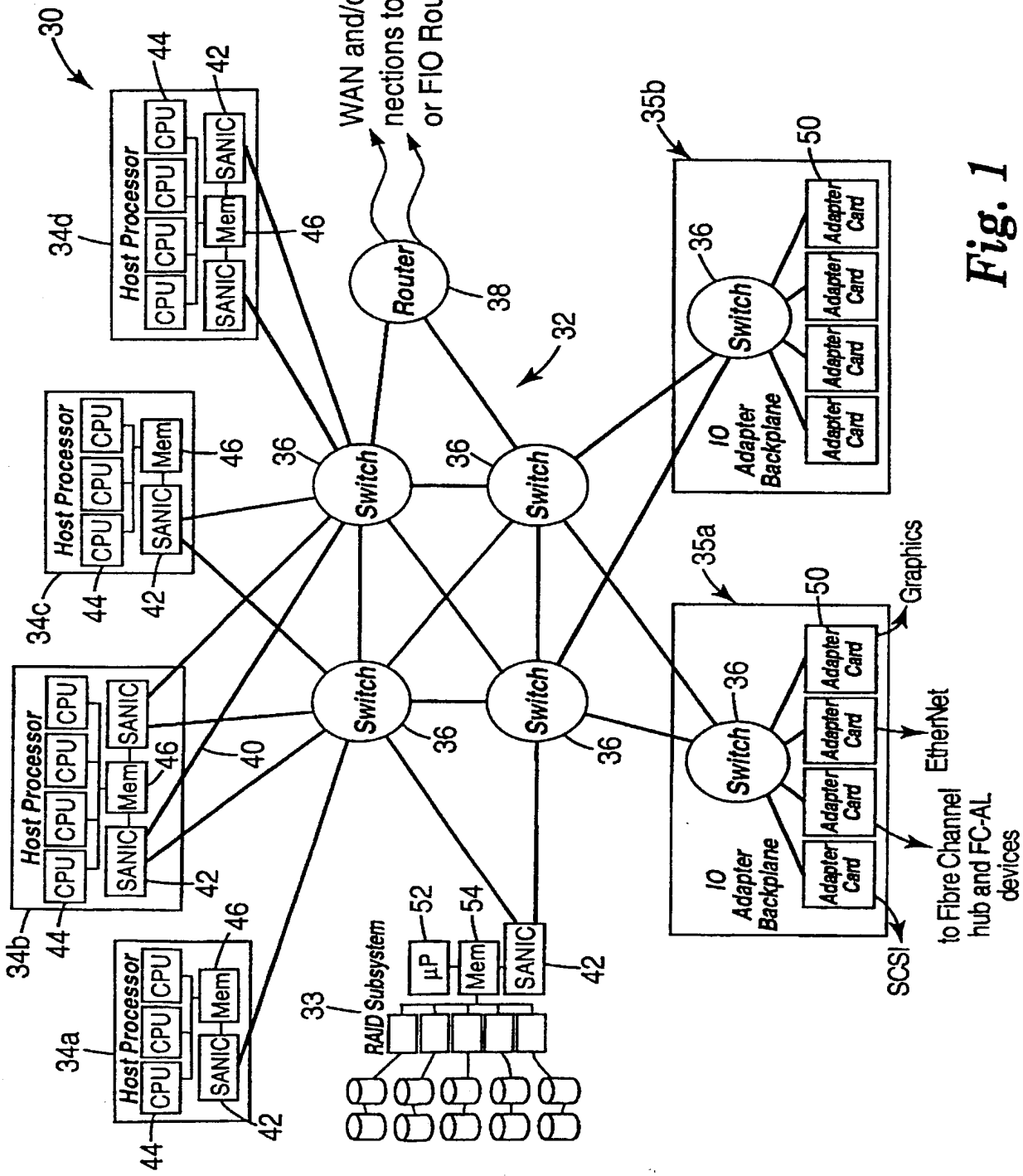

WAN and/or LAN connections to other Hosts or FIO Routers.
Router
38
Switch
36
32
Host Processor
CPU
SANIC
Mem
30
34a
34b
34c
34d
40
42
44
46
RAID Subsystem
33
µP
52
54
35a
35b
IO Adapter Backplane
Adapter Card
50
Graphics
EtherNet
SCSI
to Fibre Channel hub and FC-AL devices

Fig. 1

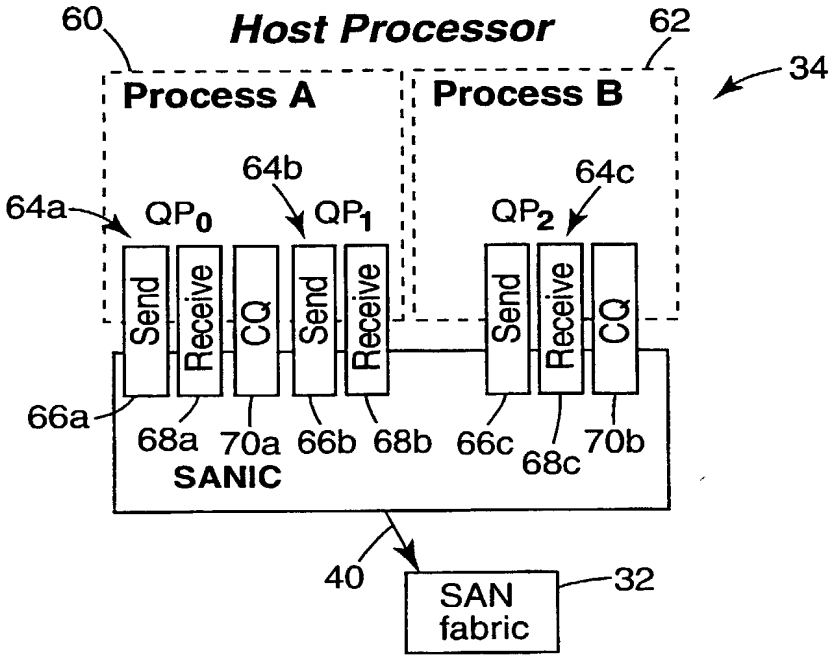

60
Host Processor
62
34
Process A
Process B
64a
64b
64c
QP0
QP1
QP2
Send
Receive
CQ
Send
Receive
Send
Receive
CQ
66a
68a
70a
66b
68b
66c
68c
70b
SANIC
40
SAN fabric
32

Fig. 2

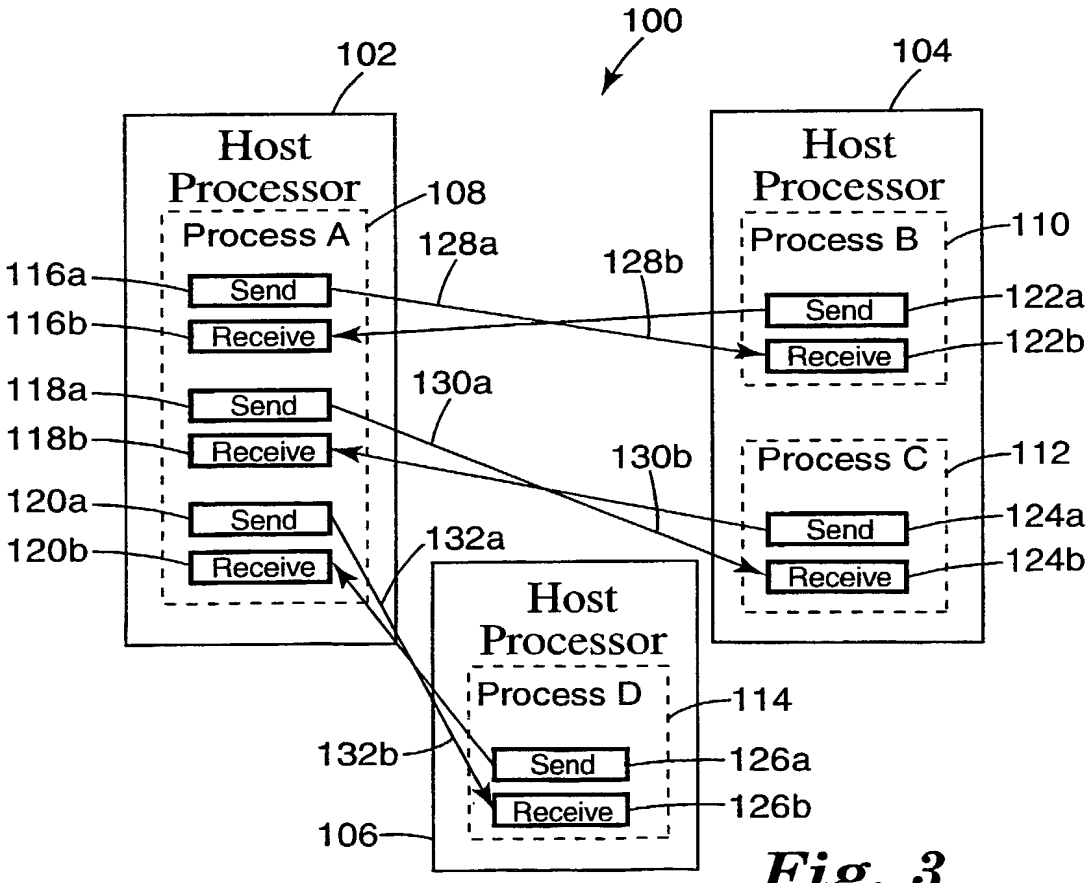

100
102
104
Host Processor
Process A
108
Host Processor
Process B
110
128a
128b
116a
116b
Send
Receive
Send
Receive
122a
122b
118a
118b
Send
Receive
130a
130b
Process C
112
120a
120b
Send
Receive
Send
Receive
124a
124b
132a
Host Processor
Process D
114
132b
Send
Receive
126a
126b
106

Fig. 3

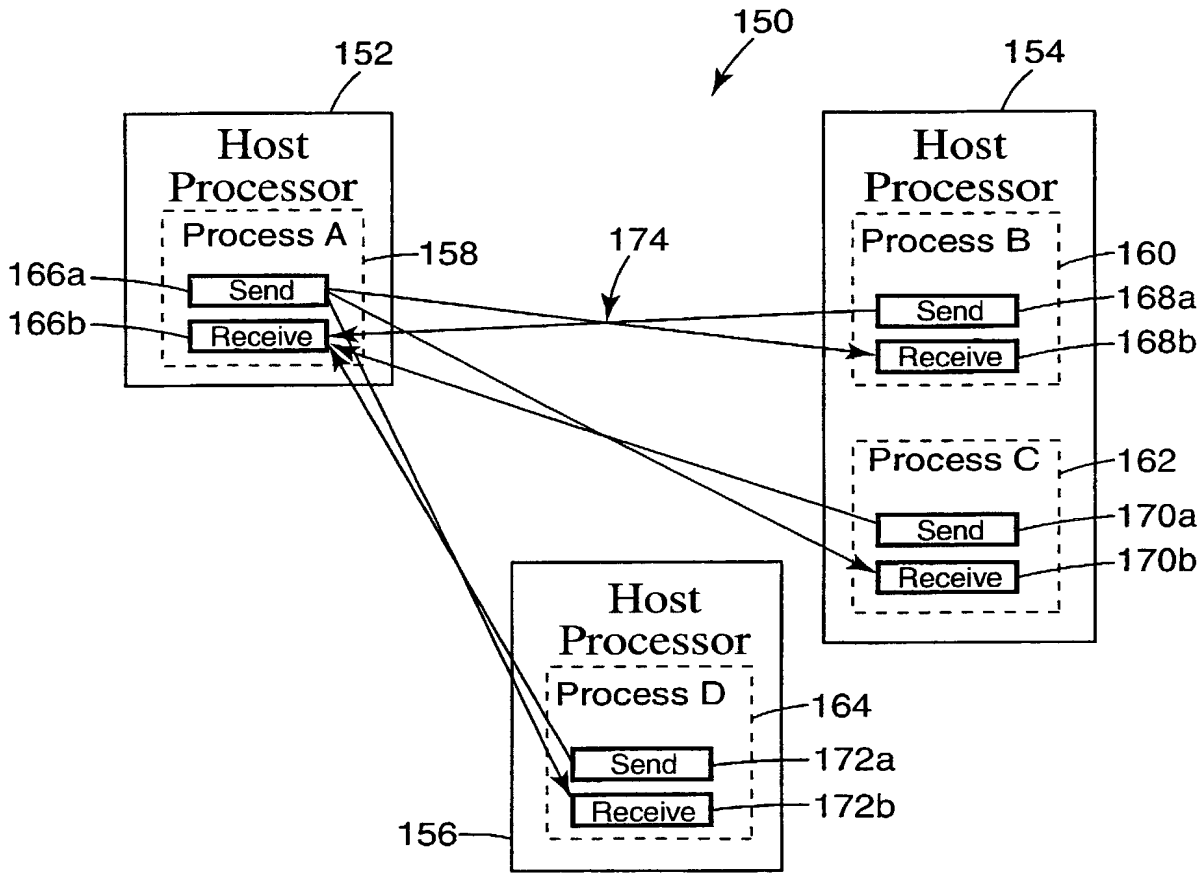

150
152
154
Host Processor
Process A
158
174
Process B
160
166a
Send
168a
166b
Receive
Send
168b
Receive
Process C
162
Send
170a
Receive
170b
Host Processor
Process D
164
Send
172a
Receive
172b
156

Fig. 4

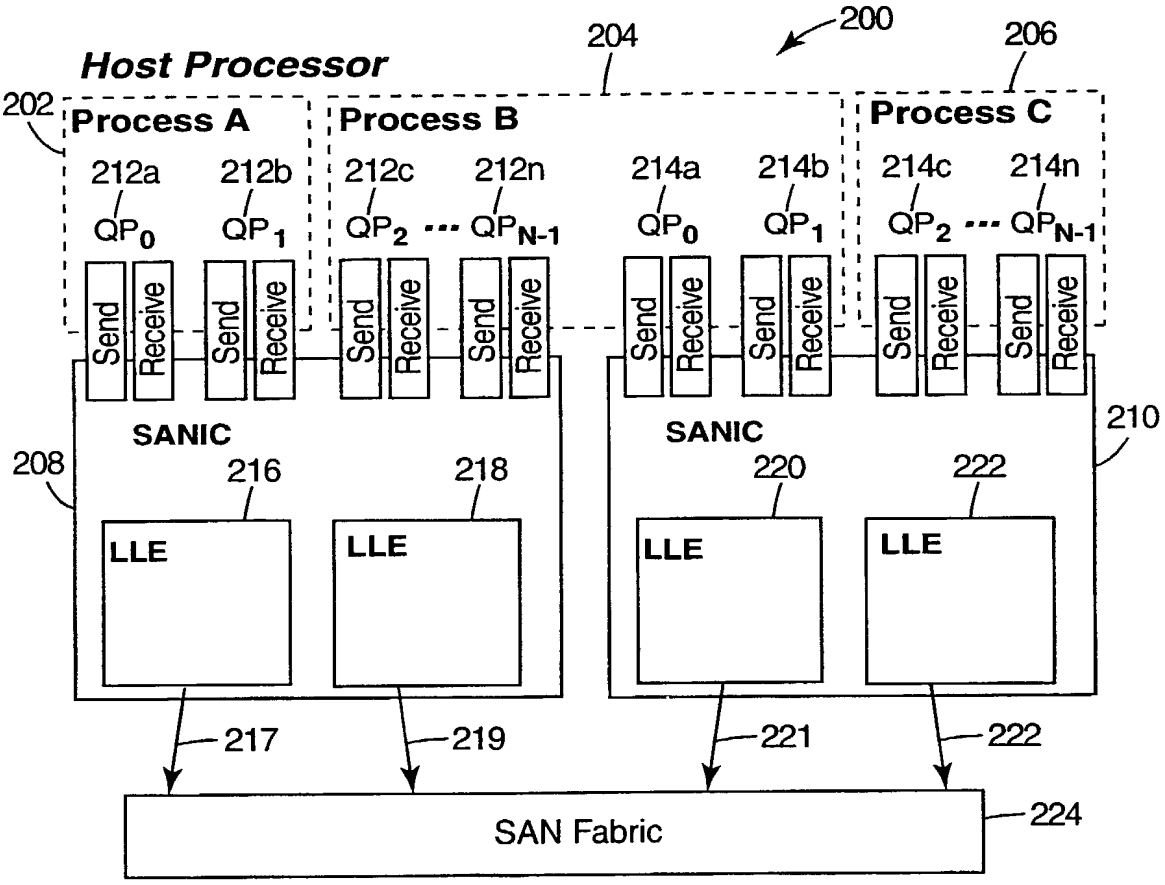

Host Processor
200
202
204
206
Process A
Process B
Process C
212a
212b
212c
212n
214a
214b
214c
214n
QP0
QP1
QP2
QPN-1
Send
Receive
SANIC
208
210
216
218
220
222
LLE
217
219
221
222
SAN Fabric
224

Fig. 5

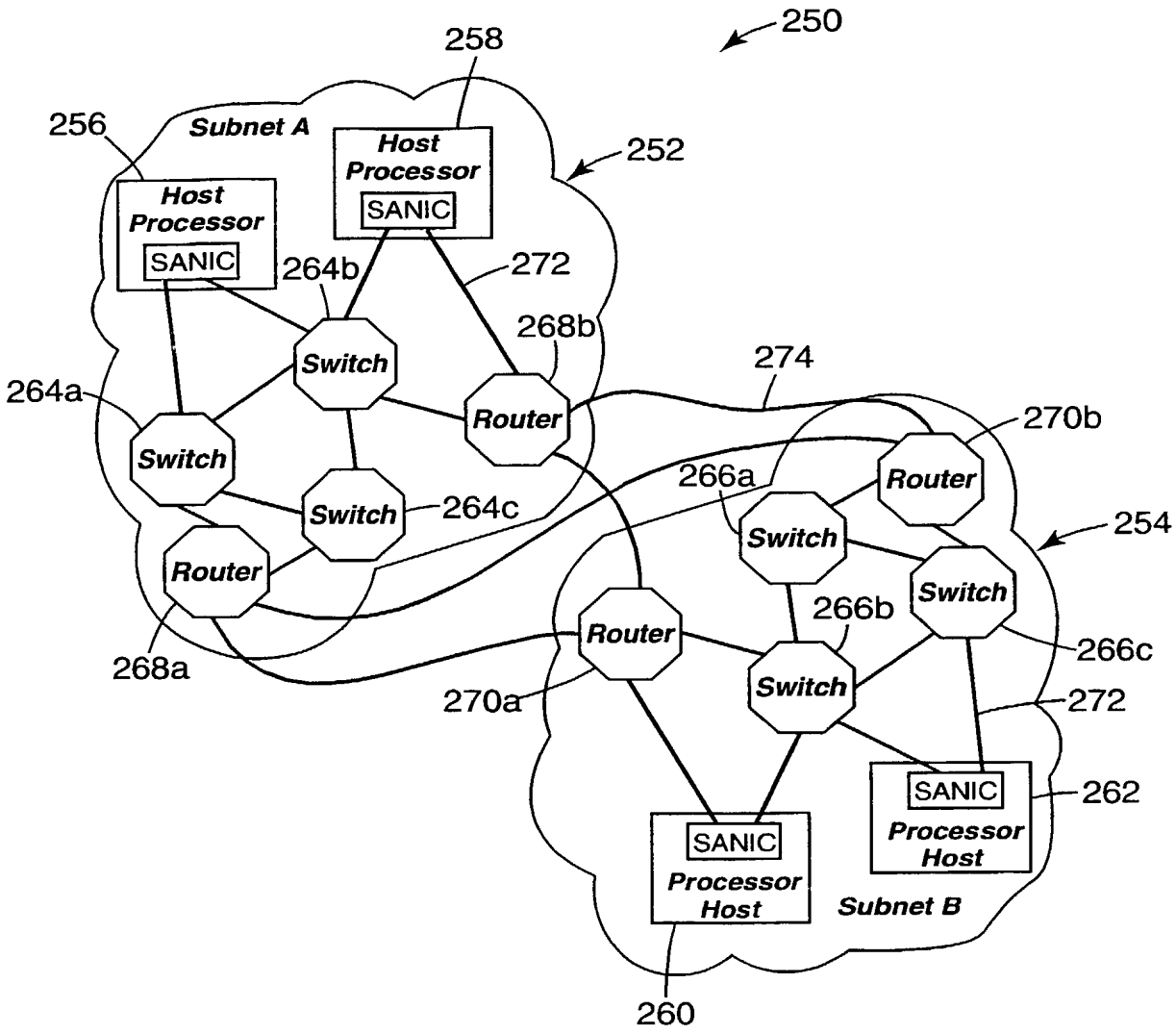

250
258
256
Subnet A
Host Processor
SANIC
252
Host Processor
SANIC
264b
272
268b
274
264a
Switch
Router
270b
Switch
266a
Router
Switch
264c
254
Switch
Router
266b
Switch
Router
266c
268a
270a
272
SANIC
Processor Host
262
SANIC
Processor Host
Subnet B
260

Fig. 6

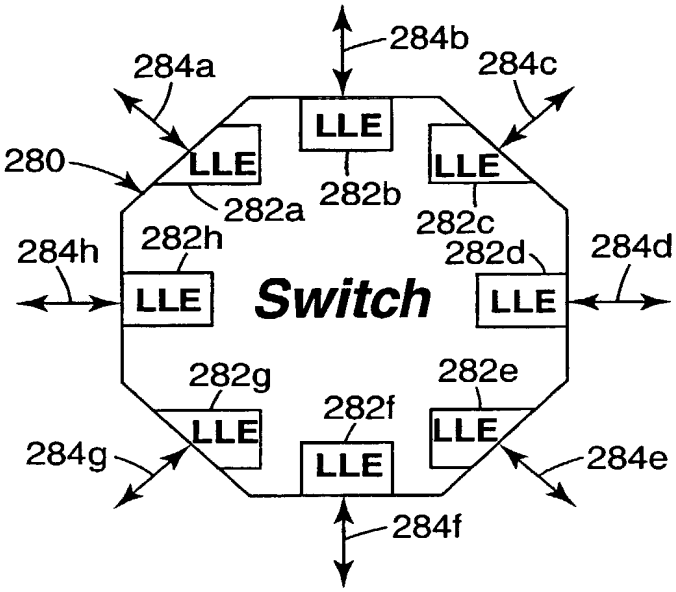

284b
284a
284c
LLE
LLE
LLE
280
282a
282b
282c
282h
282d
284h
284d
LLE
Switch
LLE
282g
282e
282f
LLE
LLE
LLE
284g
284e
284f

Fig. 7

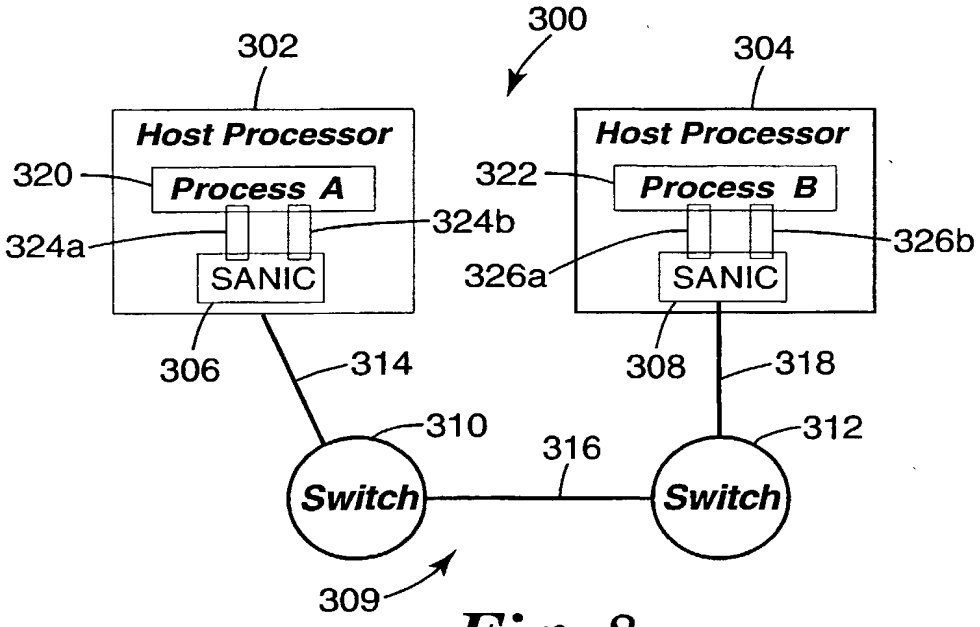

300
302
304
Host Processor
Host Processor
320
Process A
322
Process B
324a
324b
326a
326b
SANIC
SANIC
306
308
314
318
310
312
316
Switch
Switch
309

Fig. 8

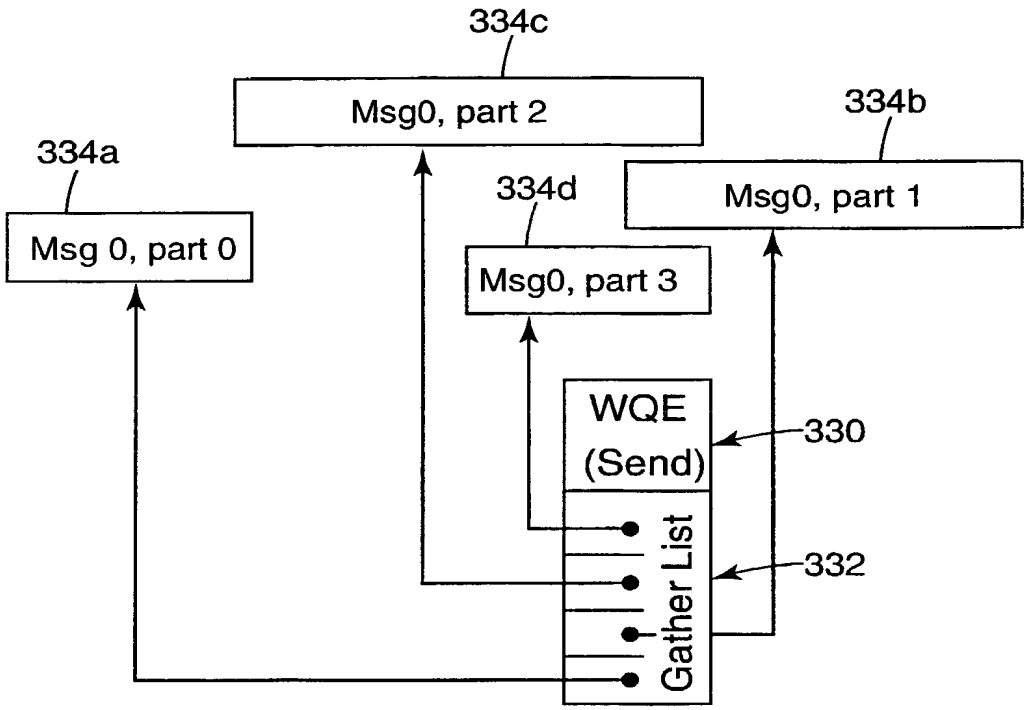

334c
Msg0, part 2
334b
334a
Msg0, part 1
334d
Msg 0, part 0
Msg0, part 3
WQE
(Send)
330
Gather List
332

Fig. 9A

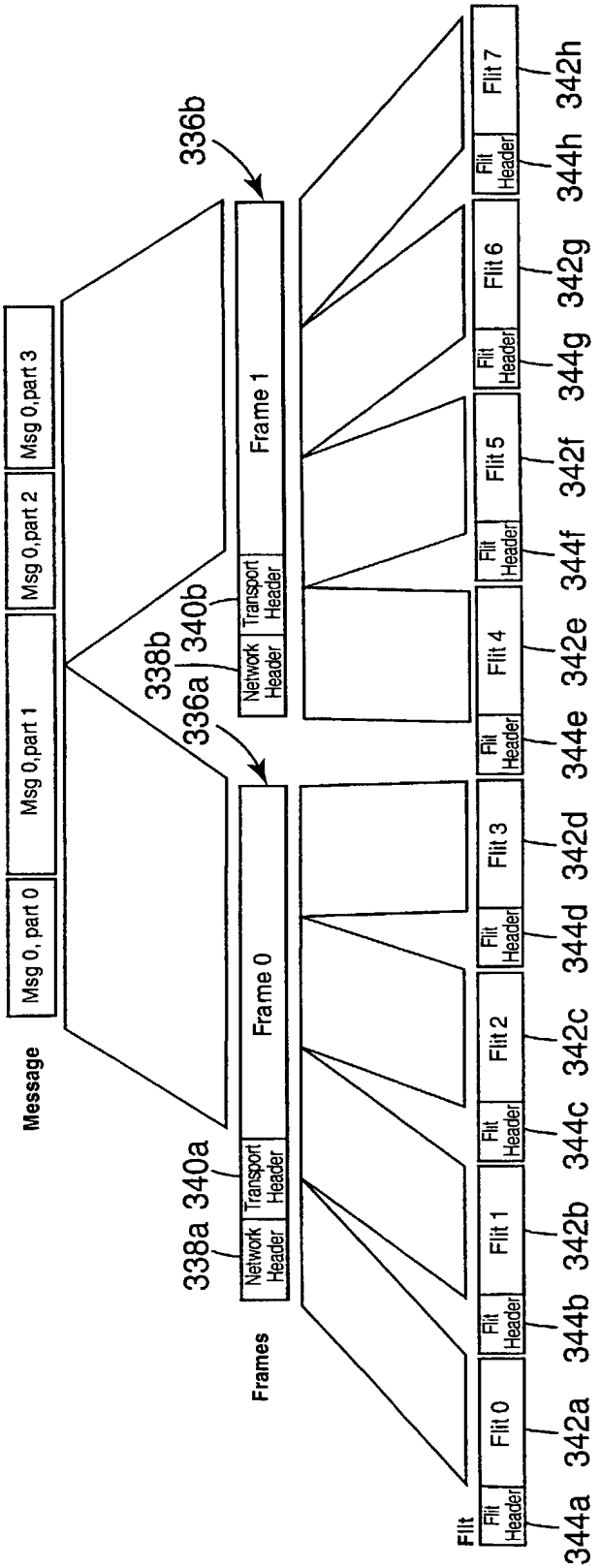

Message
Msg 0, part 0
Msg 0,part 1
Msg 0,part 2
Msg 0,part 3
Frames
Frame 0
Frame 1
336a
336b
338a
338b
340a
340b
Network Header
Transport Header
Flit
Flit Header
Flit 0
Flit 1
Flit 2
Flit 3
Flit 4
Flit 5
Flit 6
Flit 7
342a
342b
342c
342d
342e
342f
342g
342h
344a
344b
344c
344d
344e
344f
344g
344h

Fig. 9B

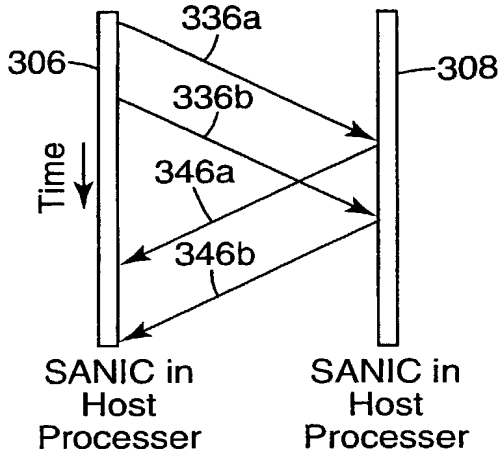


Fig. 10A

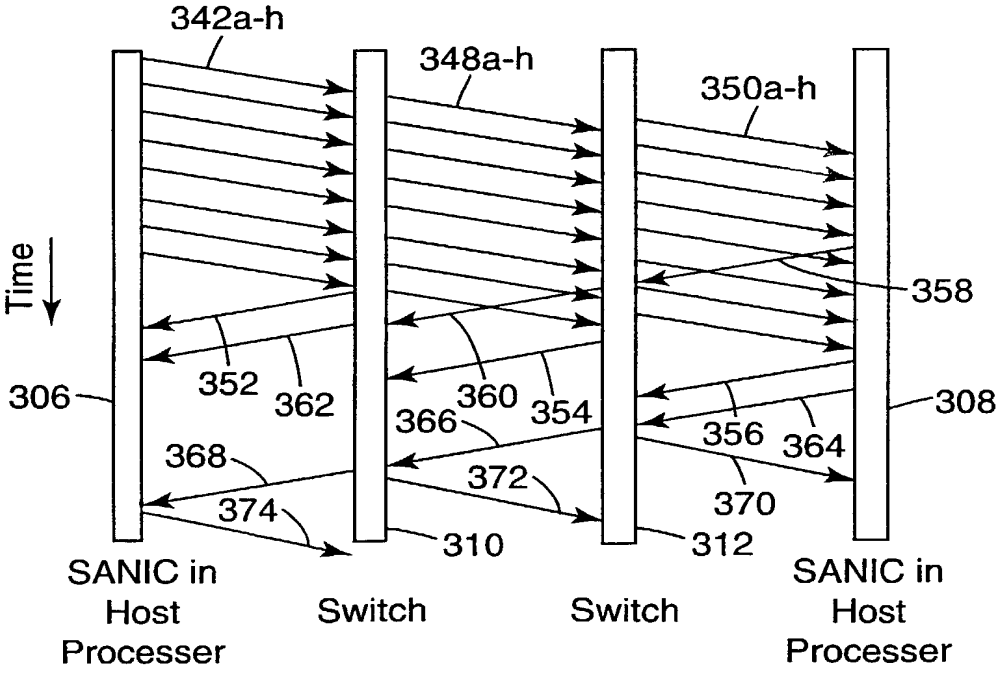


Fig. 10B

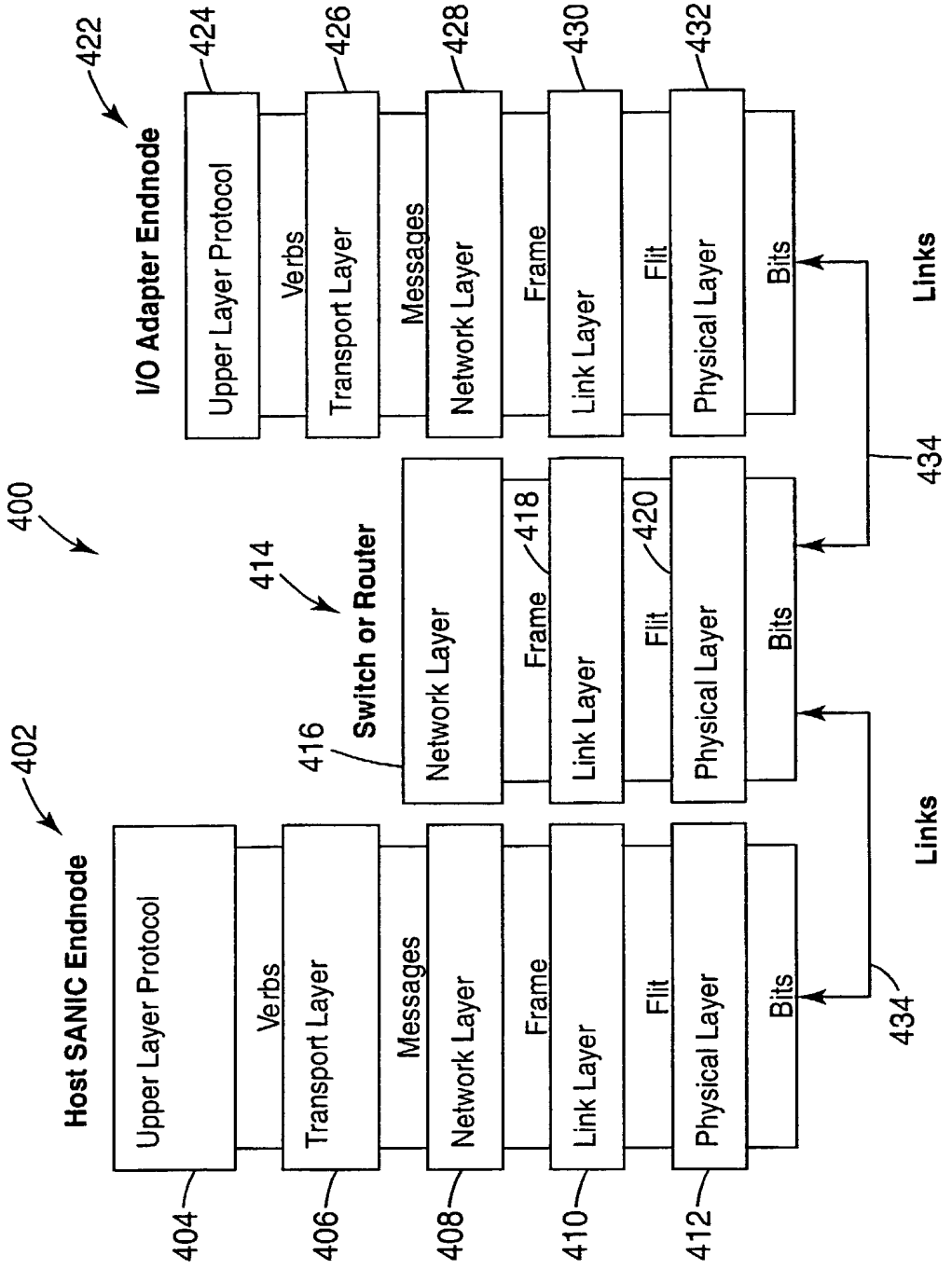

400
402
Host SANIC Endnode
Upper Layer Protocol
Verbs
Transport Layer
Messages
Network Layer
Frame
Link Layer
Flit
Physical Layer
Bits
404
406
408
410
412
414
Switch or Router
416
418
420
422
I/O Adapter Endnode
424
426
428
430
432
434
Links

Fig. 11

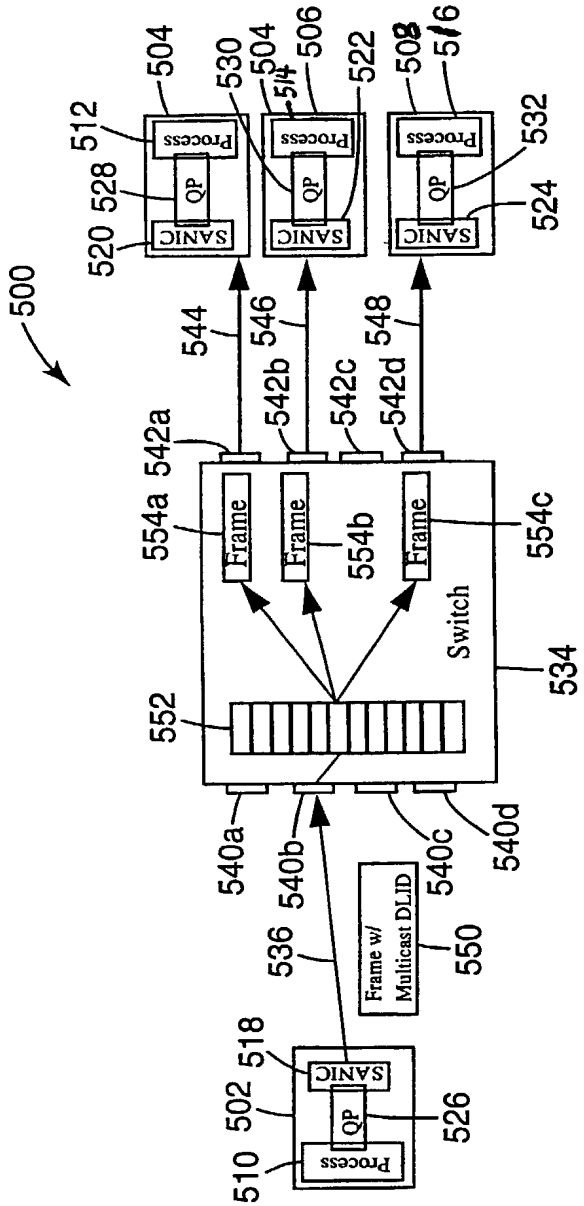

500
504
530
504
506
522
508
516
512
514
532
528
524
520
544
546
548
542a
542b
542c
542d
554a
554b
554c
Frame
Frame
Frame
Switch
534
552
540a
540b
540c
540d
536
Frame w/ Multicast DLID
550
518
502
526
510
SANIC
QP
Process

Fig. 12

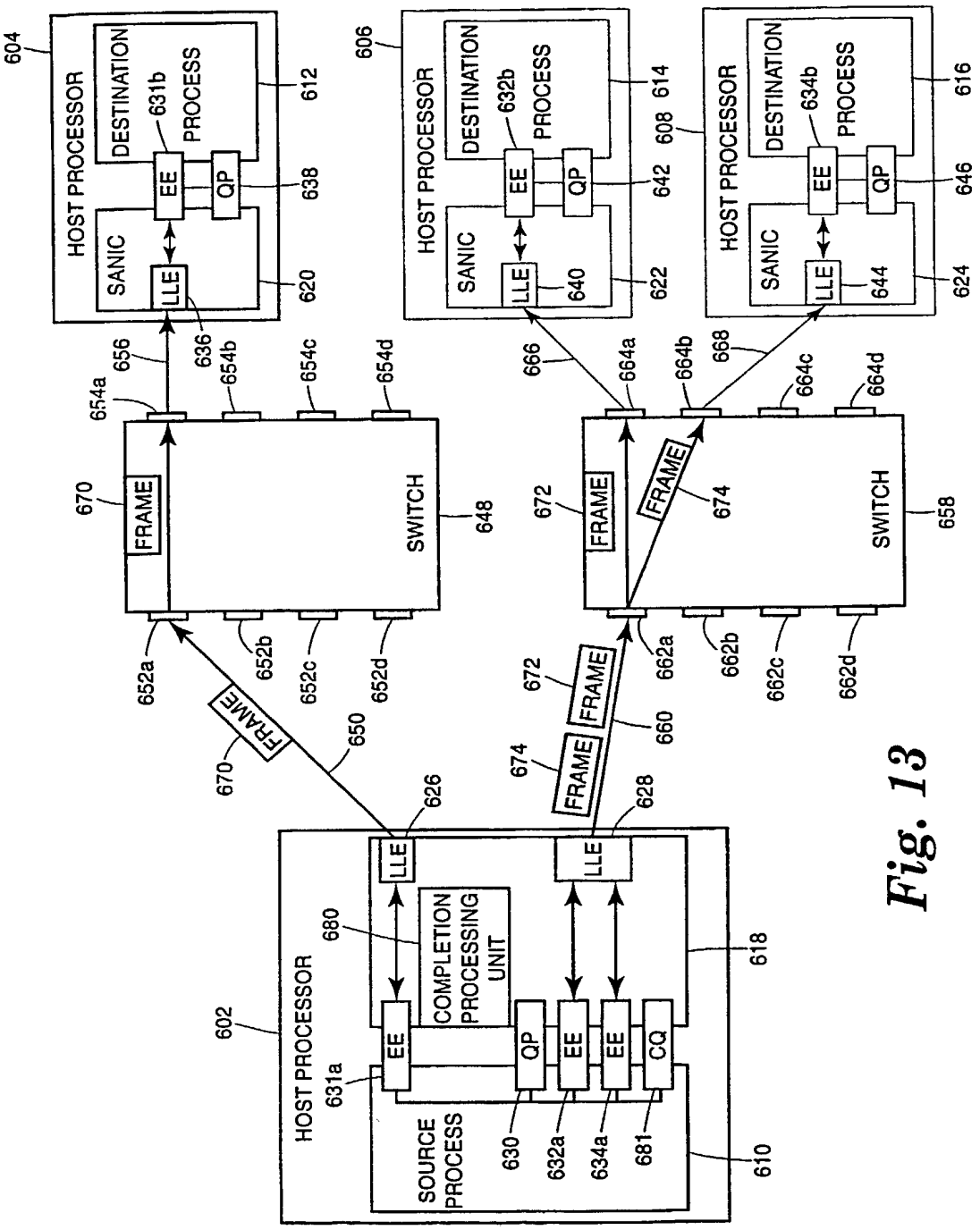

HOST PROCESSOR
DESTINATION
PROCESS
SANIC
EE
QP
LLE
SWITCH
FRAME
COMPLETION PROCESSING UNIT
CQ
SOURCE PROCESS
604
606
608
610
612
614
616
618
620
622
624
626
628
630
631a
631b
632a
632b
634a
634b
636
638
640
642
644
646
648
650
652a
652b
652c
652d
654a
654b
654c
654d
656
658
660
662a
662b
662c
662d
664a
664b
664c
664d
666
668
670
672
674
680
681

Fig. 13

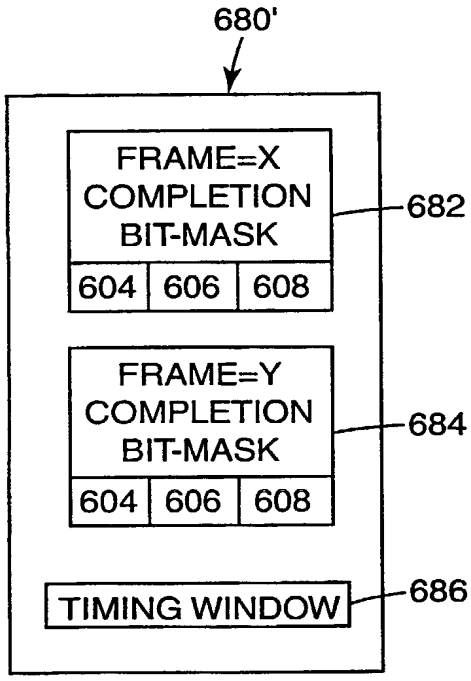

680'
FRAME=X COMPLETION BIT-MASK
604
606
608
682
FRAME=Y COMPLETION BIT-MASK
604
606
608
684
TIMING WINDOW
686

Fig. 14

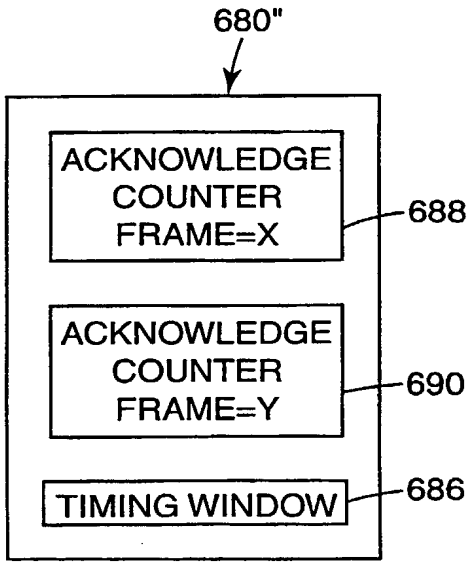

680"
ACKNOWLEDGE COUNTER FRAME=X
688
ACKNOWLEDGE COUNTER FRAME=Y
690
TIMING WINDOW
686

Fig. 15

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION

ATTORNEY DOCKET NO. 10003628

As a below named inventor, I hereby declare that:

My residence/post office address and citizenship are as stated below next to my name;

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled:

RELIABLE MULTI-UNICAST

the specification of which is attached hereto unless the following box is checked:

(X) was filed on Nov. 26, 2001 as US Application No. or PCT International Application Number 09/980,761 and was amended on 11/26/01 (if applicable).

I hereby state that I have reviewed and understood the contents of the above-identified specification, including the claims, as amended by any amendment(s) referred to above. I acknowledge the duty to disclose all information which is material to patentability as defined in 37 CFR 1.56.

Foreign Application(s) and/or Claim of Foreign Priority

I hereby claim foreign priority benefits under Title 35, United States Code Section 119 of any foreign application(s) for patent or inventor(s) certificate listed below and have also identified below any foreign application for patent or inventor(s) certificate having a filing date before that of the application on which priority is claimed:

COUNTRY	APPLICATION NUMBER	DATE FILED	PRIORITY CLAIMED UNDER 35 U S C 119
PCT	PCT/US00/14250	May 24, 2000	YES X NO. ___
			YES ___ NO ___

Provisional Application

I hereby claim the benefit under Title 35, United States Code Section 119(e) of any United States provisional application(s) listed below:

APPLICATION NUMBER	FILING DATE
60/135,664	5/24/99
60/154,150	9/15/99

U. S. Priority Claim

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

APPLICATION NUMBER	FILING DATE	STATUS (patented/pending/abandoned)

POWER OF ATTORNEY:

As a named inventor, I hereby appoint the following attorney(s) and/or agent(s) to prosecute this application and transact all business in the Patent and Trademark Office connected therewith:

Customer Number 022879

Place Customer Number Bar Code Label here

Send Correspondence to:	Direct Telephone Calls To:
HEWLETT-PACKARD COMPANY Intellectual Property Administration P.O. Box 272400 Fort Collins, Colorado 80527-2400	Patrick G. Billig (612) 573-2003

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Full Name of Inventor: Michael R. Krause Citizenship: U.S.A.

Residence: 220 Hawk Ridge Road, Boulder Creek, CA 95006

Post Office Address: Same as above

Michael R. Krause
Inventor's Signature

3/25/2002
Date

DECLARATION AND POWER OF ATTORNEY FOR PATENT APPLICATION (continued)

ATTORNEY DOCKET NO. 10003628

Full Name of # 2 joint inventor: Renato J. Recio

Citizenship: U.S.A.

Residence: 11400 Burnette Road, Austin, TX 78758

Post Office Address: Same as above

Inventor's Signature

Date 3/25/02

Full Name of # 3 joint inventor:

Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date

Full Name of # 4 joint inventor:

Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date

Full Name of # 5 joint inventor:

Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date

Full Name of # 6 joint inventor:

Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date

Full Name of # 7 joint inventor:

Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date

Full Name of # 8 joint inventor:

Citizenship:

Residence:

Post Office Address:

Inventor's Signature

Date